FASTBUS Smart Crate Controller (FSCC)
Hardware Manual

Version PC4b

Abstract:

The FSCC is a simple FASTBUS readout controller designed for low occupancy front-end modules. It performs most FASTBUS master operations, but is not intended to be a "general-purpose" FASTBUS master. The module features a Motorola 68020 processor with a "Thin-wire" Ethernet port, allowing an imbedded operating system to be installed. A FIFO memory buffers front-end data read in through FASTBUS. The FSCC then adds a leading word count, and transmits the data out of a 32-bit port on the FASTBUS Auxiliary connector. A "personality" card (which can contain active components) is installed in the FASTBUS Auxiliary backplane, to convert the 32-bit parallel TTL data into any desired format for data transmission. Information on current FSCC personality cards is included as an appendix of this document. This document supersedes HN96, but does not replace it. HN96 applies to FSCC's of version PC4 and earlier.

Principal Authors of Original Document:
Mark Bowden,
Gustavo Cancelo,
Richard Kwarciany,
John Urish

PC4/PC4a/PC4b Revisions:
Richard Kwarciany

Contributing Authors:
Mark Bernett,
Robert Forster,
James Franzen,
Oscar Trevizo

November 1, 1995

HN 136

DISCLAIMER NOTICE

Table of Contents

Table of Figures

Tables

Blank

1. General Information

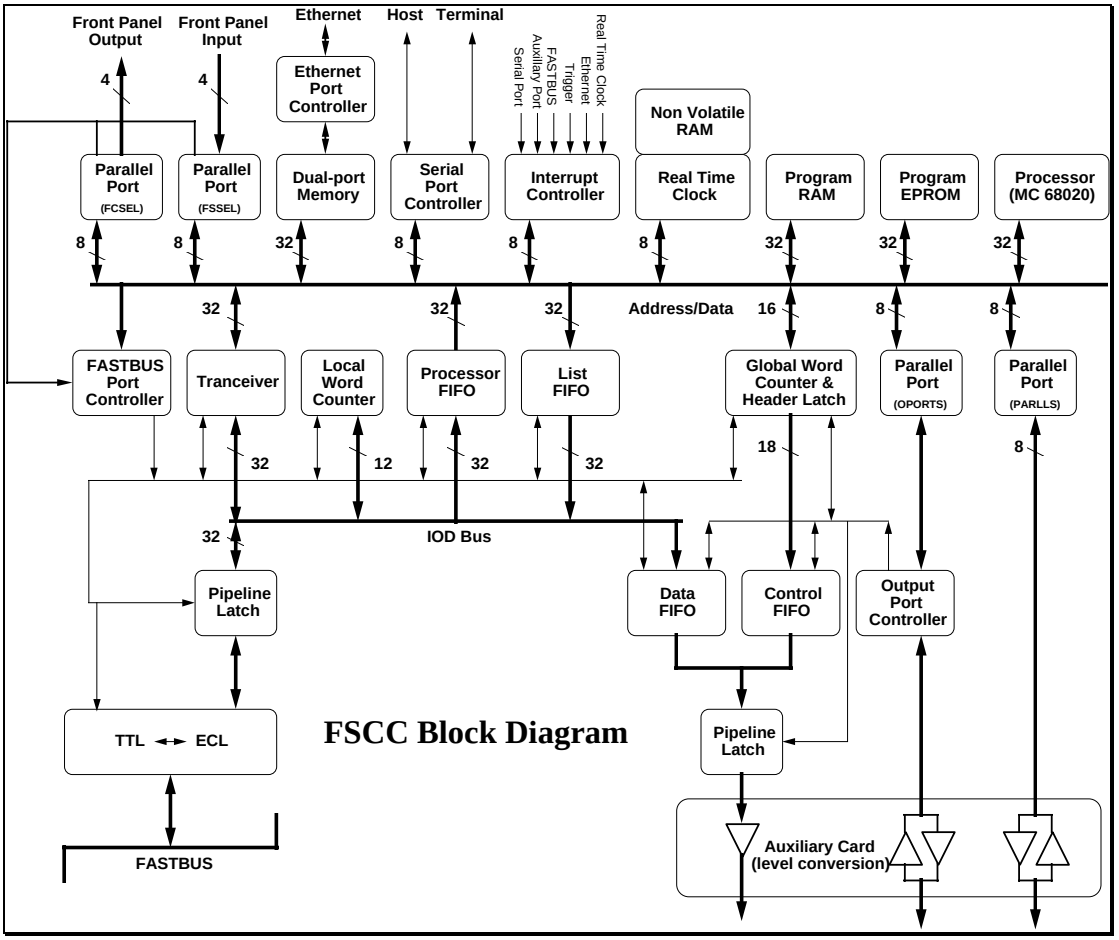


Figure 1 FSCC Block Diagram

1.1 Purpose

The FSCC was designed as a simple readout controller for low occupancy front-end modules. It performs most basic FASTBUS operations but was not intended to be a "general-purpose" FASTBUS master.

A Motorola 68020 processor is used to control operation of the module and any features which are not time-critical have been allocated to software.

The original design goals were as follows;

a) a typical readout time of 1 µsec for a single slave module with a few words of data, including primary address and address release,

b) ability to execute most standard FASTBUS Master operations, and

c) design simplicity such that a working prototype module could be assembled in 6-9 months.

1.2 Standard Bus Connections (FASTBUS)

The FSCC operates as a master on both the FASTBUS crate segment port and the (non-FASTBUS) auxiliary port. It also supports FASTBUS slave operations on the crate segment, but at a very reduced rate. All slave operations are a function of software.

1.3 Packaging

The FSCC is a single-width FASTBUS module containing approximately 100 integrated circuits. Physical dimensions of the module are per the FASTBUS specification. The PC4b version also has a small child board containing the Trigger FIFO and associated logic, and NIM level converters, near the module front panel.

1.3.1 Module Pinout (Backplane Connections)

a) FASTBUS Crate Segment (130 pin FASTBUS standard connector)
Refer to the FASTBUS specification.

b) Auxiliary Port (195 pin FASTBUS standard 3 row connector), see Table 4.

1.3.2 Front Panel

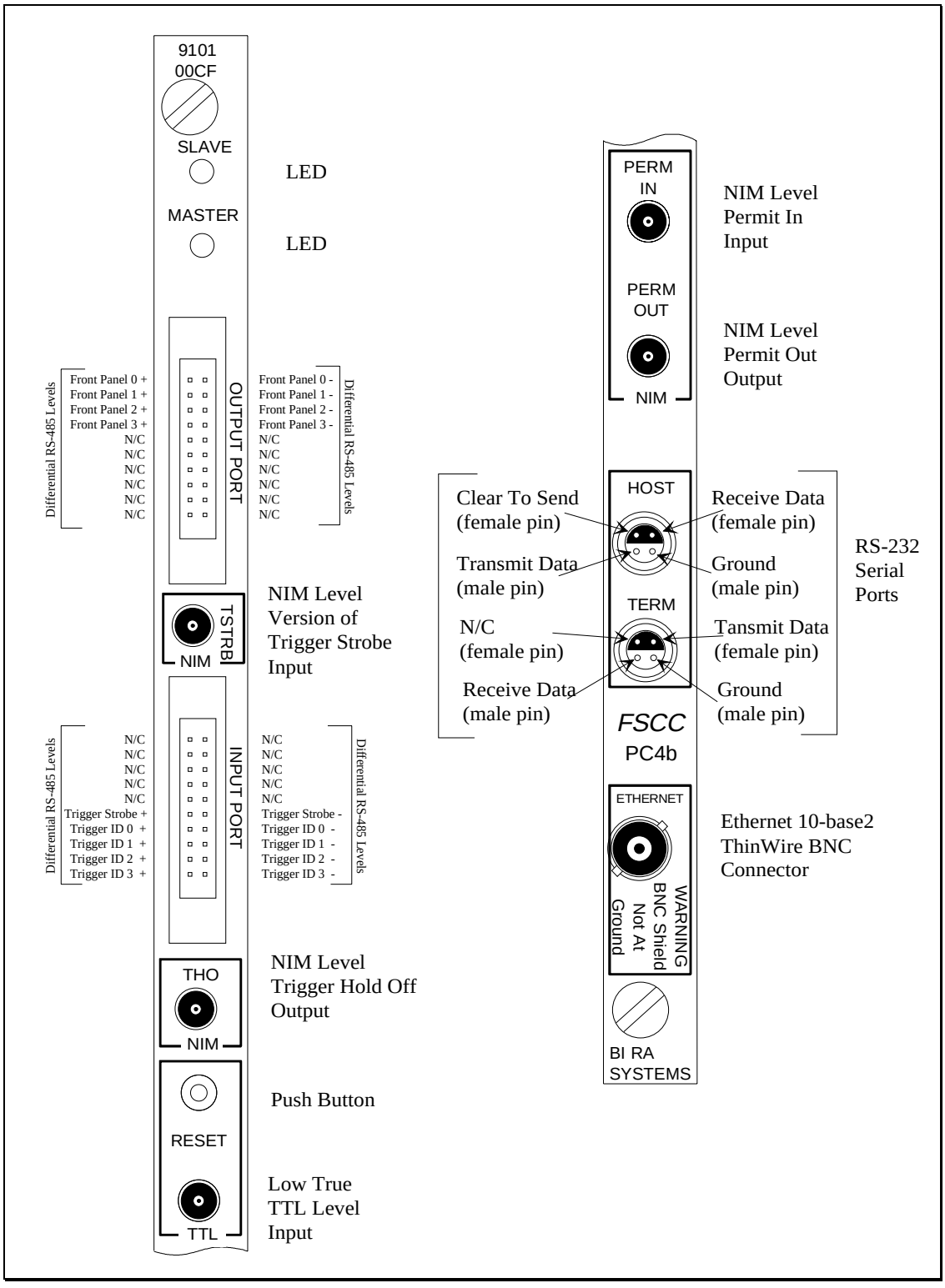


Figure 2 FSCC Front Panel

FASTBUS Slave Yellow LED. Indicates that the FSCC has been addressed as a slave on FASTBUS.

FASTBUS Master Green LED. Indicates that the FSCC is Master on FASTBUS.

Front Panel Output: The Front Panel Output Port consists of four latched differential RS485 pairs, which are driven by the processor via a parallel port. These are user defined.

Table 1 Front Panel Output Port Pinout

Pin#	
1	Front Panel 0 +
2	Front Panel 0 -
3	Front Panel 1 +
4	Front Panel 1 -
5	Front Panel 2 +
6	Front Panel 2 -
7	Front Panel 3 +
8	Front Panel 3 -
9-20	Reserved

NIM Trigger Strobe NIM level version of Trigger Strobe input. NIM Trigger Strobe is logically ORed with the RS-485 version of Trigger Strobe on the front panel Input Port Connector. (See Input Port below).

Input Port The front panel Input Port (DART Trigger Link Input) consists of five differential RS 485 pairs. Four of the differential pairs are the four bits of the Trigger ID. The fifth differential pair is the RS-485 Trigger Strobe Input. The RS-485 Trigger Strobe input is logically ORed with the NIM level Trigger Strobe input. The 4 bit Trigger ID is clocked into a 64 word FIFO (First In, First Out) memory by the leading edge (high to low transition) of the Trigger Strobe Input. The purpose of this FIFO is to allow the FSCC to be used with front end modules which are capable of buffering more than one event's worth of data. The Trigger FIFO allows the FSCC to queue as many as 64 readout triggers. The Trigger ID of the next event to be output, can be read by the processor through a parallel port. If enabled in the H&C Control register, the four trigger ID bits are passed automatically from the Trigger FIFO into the Header/Word Count word of the Data Output Port. Executing a FASTBUS End Of Event instruction clocks the next Trigger ID out of the Trigger FIFO. The Trigger FIFO's Output Ready signal generates a processor interrupt if enabled. Output Ready gives an active edge when the Trigger FIFO goes from empty to not empty, and whenever an End Of Event instruction is executed. The Trigger ID should be valid for at least 100 nsec prior to the leading edge of the Trigger Input Strobe and remain valid for at least 100 nsec after the leading edge of the strobe. The Trigger Strobe pulse width should be at least 100 ns.

Note: Resetting the Trigger FIFO when the Trigger Strobe input is active (- input high, + input low) causes a superfluous word to be clocked into the Trigger FIFO. This will cause a TSTRB interrupt if enabled. A floating (unconnected) RS-485 Trigger Strobe Input is always interpreted as inactive. The Trigger FIFO is reset whenever the Header and Counter is reset.

Table 2 Front Panel Input Port Pinout

Pin#	
1-10	Reserved
11	Trigger Strobe +
12	Trigger Strobe -
13	Trigger ID0 +
14	Trigger ID0 -
15	Trigger ID1 +
16	Trigger ID1 -
17	Trigger ID2 +
18	Trigger ID2 -
19	Trigger ID3 +
20	Trigger ID3 -

Trigger Hold Off (THO) This NIM level output can be configured via a jumper block on the child board, to go true during one of the following four conditions:

1) The Trigger FIFO is Almost Full (True when 56 or more Trigger Strobes are queued).
2) The Trigger FIFO is Half Full (True when 32 or more Trigger Strobes are queued).
3) An End Of Event FASTBUS instruction has been executed (a 100ns pulse).
4) The Trigger FIFO is not empty (True when 1 or more Trigger Strobes are queued).

THO is also driven true while the FSCC's Data FIFO is half full. THO will latch true if the Data FIFO becomes full. The THO latch is reset by resetting the Header and Counter.

Permit_In/Permit_Out: Serial "daisy-chain" signals for FSCC auxiliary port bussing, LEMO connectors. NIM level (150ns) pulse. When the Output Port is configured to be either middle or last in a permit chain, the Output Port begins outputting data when Permit In is received. Permit Out is generated when the event has been completely output. When the Output Port is configured to be first in a permit chain, the Output Port begins outputting data without a Permit In the first time after configuration. Each subsequent time, the Output Port waits for Permit In.

Reset Push-button Hard processor reset.

Remote Reset Hard processor reset. This is an active low TTL input. Shorting the connector, inserting a 50 ohm terminator, or applying a TTL low will cause a reset. This input may be Daisy-Chained.

Serial Ports RS-232 signal levels (4 pin LEMO connectors). One connector for Host communication, and one for Terminal communication.

Table 3 Front Panel Serial Port Connector Pinouts

Terminal port	
Pin #	Function
1	TXD (transmit data)
2	(no connection)
3	RXD (receive data)
4	Ground
Host port (Null Modem)	
Pin #	Function
1	RXD (receive data)
2	CTS (+10V Reference)
3	TXD (transmit data)
4	Ground

Ethernet Port: IEEE 802.3/Cheapernet (10BASE2) LAN Standard, Isolated BNC Connector.

Table 4 Auxiliary Port Connector Pinout

Facing REAR of Module

PIN	Row A	Row B	Row C
1	No Connection	No Connection	No Connection
2	No Connection	No Connection	No Connection
3	No Connection	No Connection	No Connection
4	No Connection	AC00	No Connection
5	No Connection	AC01	No Connection
6	No Connection	AC02	No Connection
7	No Connection	AC03	No Connection
8	No Connection	AC04 (EOR Out Enable)	No Connection
9	No Connection	No Connection	No Connection
10	No Connection	No Connection	No Connection
11	No Connection	AC05	No Connection
12	-5.2 Volt Supply	AC06	+5.0 Volt Supply
13	No Connection	AC07	No Connection
14	No Connection	AC08 (Data Strobe)	No Connection
15	No Connection	AC09 (End Of Record)	No Connection
16	No Connection	D00	No Connection
17	No Connection	D01	No Connection
18	No Connection	D02	No Connection
19	No Connection	D03	No Connection
20	GND	D04	No Connection
21	No Connection	D05	No Connection
22	GND	D06	GND
23	No Connection	D07	No Connection
24	No Connection	D08	No Connection
25	No Connection	D09	No Connection
26	No Connection	D10	No Connection
27	No Connection	D11	No Connection
28	No Connection	D12	No Connection
29	No Connection	D13	No Connection
30	No Connection	D14	No Connection
31	No Connection	D15	No Connection
32	-5.2 Volt Supply	D16	GND
33	No Connection	D17	No Connection
34	No Connection	D18	No Connection
35	No Connection	D19	No Connection
36	No Connection	D20	No Connection
37	No Connection	D21	No Connection
38	No Connection	D22	No Connection
39	No Connection	D23	No Connection
40	No Connection	D24	No Connection
41	No Connection	D25	No Connection
42	No Connection	D26	No Connection
43	+5.0 Volt Supply	D27	GND
44	No Connection	D28	GND

PIN	Row A	Row B	Row C
45	No Connection	D29	No Connection
46	No Connection	D30	No Connection
47	No Connection	D31	No Connection
48	No Connection	No Connection	No Connection
49	No Connection	No Connection	No Connection
50	No Connection	No Connection	No Connection
51	No Connection	No Connection	No Connection
52	No Connection	No Connection	No Connection
53	GND	No Connection	+5.0 Volt Supply
54	No Connection	No Connection	No Connection
55	No Connection	No Connection	No Connection
56	No Connection	No Connection	No Connection
57	No Connection	No Connection	No Connection
58	No Connection	AC10 (Wait)	No Connection
59	No Connection	AC11 (Data Out Enable)	No Connection
60	No Connection	AC12 (Strobe Out Enable)	No Connection
61	No Connection	No Connection	No Connection
62	No Connection	No Connection	No Connection
63	GND	No Connection	-5.2 Volt Supply
64	No Connection	GND	No Connection
65	No Connection	-2.0 Volt Supply	No Connection

1.4 Power Requirements

Total power dissipation is approximately 40 watts drawn from the three supplies listed in Table 5 Power Supply RequirementsTable 5.

Table 5 Power Supply Requirements

Supply	*Current Draw*	*Fuse*
+5.0 volts	6 amps	10 Amp
-5.2 volts	2 amp	5 Amp
-2.0 volts	0.2 amps	1 Amp

2. Theory of Operation and Operating Modes

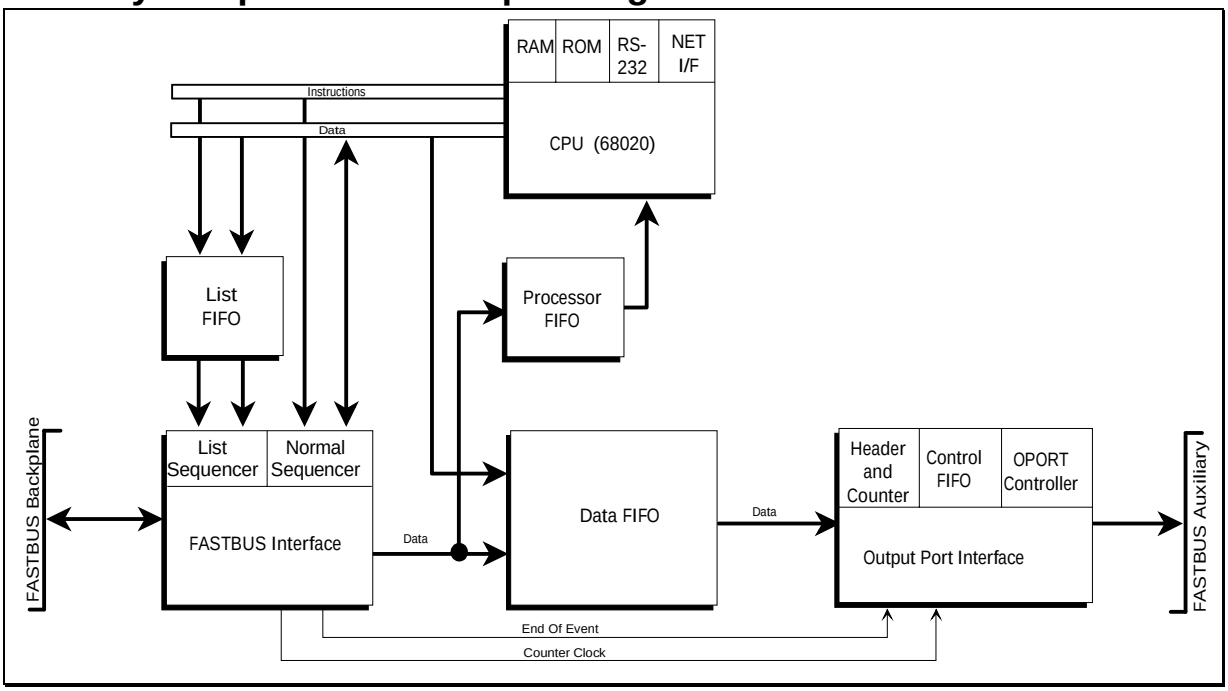


Figure 3 FSCC Functional Block Diagram

Figure 1 shows a general block diagram of the FSCC. Figure 3 shows a functional block diagram. The individual blocks are explained in detail below.

2.1 Basic Operation

All module operations are controlled by memory-mapped instructions from the **68020 Processor**.

Low level management of the various ports is handled by dedicated controllers. In the case of the **Serial** and **Ethernet Ports**, the controllers are commercial integrated circuits. Controllers for the **FASTBUS Port** (FPORT) and Auxiliary **Output Port** (OPORT) are implemented using PLD state machines.

To maximize throughput, high speed block transfer data from FASTBUS to the Auxiliary Output Port is routed through the **Data FIFO** which serves two functions; it 1) decouples the input and output data rates, and 2) provides buffering of one crate of front-end data for insertion of leading word counts on output. The buffer is implemented using commonly available integrated circuit FIFO's which provide a depth of 4K, 32-bit words (16K Bytes). The processor can write data directly into the Data FIFO for testing purposes.

A 512 x 32-bit (2K Bytes) **Processor FIFO** connects the FASTBUS port to the Processor. Input to this FIFO can be enabled or disabled by the Processor. When enabled, any data which is written to the Data FIFO will simultaneously be written to the Processor FIFO. The Processor FIFO provides a high-speed path through which the processor can sample data in the event stream.

A 512 x 40-bit (512 x 32-bit FASTBUS operand, and 512 x 8-bit instruction) **List FIFO** also connects the Processor to the FASTBUS data port. This FIFO is written by the Processor and contains a FASTBUS instruction list for use by the FPORT Controller (List Sequencer). Use of the List FIFO increases the speed of the readout by eliminating the overhead penalty induced by the CPU. The price paid for the speed increase is reduced FASTBUS

error reporting ability. When the OPORT controller encounters an ERROR, it simply sets the “Bad Event” bit in the OPORT Header/Word Count word, and continues.

To further improve the readout speed of the FSCC, a **Local Word Counter** is implemented. In a typical application, this counter is loaded directly from the first word of a Block or Pipelined transfer, with a fixed word count position (bits 0-11). The counter can also be preset by the processor for use with slave modules which do not supply a leading word count.

For slave modules capable of producing leading word counts, the local counter eliminates the extra delay the FPORT Controller normally incurs in checking for the SS=2 "end-of-block" condition. It allows better pipelining of the data since the controller will not be required to read beyond end-of-block and then back-up it's internal counters and pipeline registers. The Local Word Counter can also be used to produce intermediate word counts, at user-selected boundaries for insertion in the main data stream. Maximum word count is 4095.

A **Global Word Counter and Header Latch (GWC)** is also implemented. This is normally used to provide a total count of all data read from a group of modules. It is clocked by the FPORT Controller on each FASTBUS data word. The value of this counter is inserted into the output data stream along with 5 bits of header information on command of the OPORT Controller (execution of an End Of Event instruction). The FPORT Controller provides both a control and a data "end-of-event" (EOE) signal. The control EOE signal tells the H&C Controller to push the GWC along with the current header into the Control FIFO. The data EOE is put into the Data FIFO behind the last event of data so that the OPORT controller can tell where one event ends and the next begins when it clocks the data out. The word count is limited to 12 bits (4096 words). The 5 bit header field is loaded by the processor. The lower four bits of the header field can automatically contain the Trigger ID data from the Trigger FIFO if desired. Register definitions and operating modes for the header/counter latch (H&C) and OPORT controller are outlined in this document.

A small **Control FIFO** connects the Header/Word Count outputs of the H&C (Header & Counter) Controller, and Data FIFO outputs, to allow overlapping of events in the Data FIFO. This is necessary because the total word count of an event is given by the Global Word Counter, which counts the data words as they are read in. Therefore, the word count is not known until the event readout is complete. The OPORT wants to know the word count before it starts outputting an event, so that it can output the word count first (leading word count OPORT protocol). Obviously the word count cannot be inserted into the Data FIFO because it would come out last. The word count is instead pushed into a separate FIFO, the Control FIFO. When an End Of Event FPORT instruction is executed, the following happens. The FPORT inserts an End Of Event flag into the Data FIFO. It then drives the End Of Event signal to the OPORT interface. This causes the H&C Controller to push the Global Word Count value along with the Header bits into the Control. The FPORT Controller is now free to begin a new readout without concern for the data in the Data FIFO, or any event being output through the Output Port. The OPORT controller now sees that the Control FIFO has a word in it, and it pulls this word out of the Control FIFO and outputs it as soon as the Permit In token is received. It then takes the data from the Data FIFO, and outputs it one word at a time until it sees the End Of Event flag come out of the FIFO. The OPORT Controller then drives the End Of Record output (if the OPORT controller is configured to be “last” in the permit chain), then outputs the Permit Out pulse to pass the token. Note that EOE (End Of Event) is an internal FSCC signal and it should not be confused the EOR (End Of Record) which is an external signal driven by the OPORT Controller.

2.2 On-Board Processor

The processor is a Motorola 68020 running at 20 MHz. The 68020 was selected strictly on the basis of software compatibility with existing Fermilab modules.

The **PC4b** processor address map differs from FSCC versions PC4 and older. The map is identical to version PC4a. Table 6 shows the absolute address map.

Table 6 Processor Address Map

Base Address	Transfer Type	Name	Purpose
0000 0000	byte, word, long word	ROM1S*	Program EPROM (Bank 1, 2048K)
0020 0000	byte, word, long word	ROM2S*	Program EPROM (Bank 2, 2048K)
0080 0000	byte, word, long word	RAM1S*	Program RAM (Bank 1, 512K)
0088 0000	byte, word, long word	RAM2S*	Program RAM (Bank 2, 512K)
		OR	
0080 0000	byte, word, long word	RAM1S*	Program RAM (Bank 1, 2048K)
00A0 0000	byte, word, long word	RAM2S*	Program RAM (Bank 2, 2048K)
0150 0000	byte	NVDS*	Real-time clock
0150 000E	byte	NVDS*	Non-volatile memory 8K (8178 Bytes)
0160 0000	byte, word, long word	ETHS*	Ethernet Dual-Port RAM (8K Bytes)
0168 0000	long word	CAS	Ethernet Channel Attention
0170 0000	byte	UARTS*	Serial Ports (68681)
0178 0000	byte	TMRS*	Interrupt Vector/Timer Logic (MC68901)
0180 0000	byte	OPORTS*	OPORT Controller
0188 0000	byte	PARLLS*	Auxiliary Parallel Port
0190 0000	word	H&CSEL*	Header/Counter Registers
01A0 0000	byte, word, long word	LPBKS	Ethernet loopback mode set
01A8 0000	byte, word, long word	LPBKC	Ethernet loopback mode clear
0198 0000	byte, word, long word	ETHRES	Ethernet reset set
01B0 0000	byte, word, long word	ETHRESC	Ethernet reset clear
01B8 0000	long word	FB1S*	FPORT Controller (Fast cycle instructions)
01C0 0000	long word	FB2S*	FPORT Controller (Slow cycle instructions)
01C8 0000	byte	FCSEL*	FASTBUS Parallel Port 1 (MC68230)
01D0 0000	byte	FSSEL*	FASTBUS Parallel Port 2 (MC68230)
01D8 0000	long word	FIFO1S*	Processor FIFO
01E0 0000	long word	FIFO2S*	FPORT List FIFO
01E8 0000	long word	FB1S* and FB2S*	User FPORT Controller select
01F0 0000	long word	FB3S*	FPORT List Halt Request

* Active low signal.

Processor memory normally consists of 1 MByte of 0 wait state RAM, and 2MBytes of 1 wait state EPROM, but it is possible to install up to 4MBytes of EPROM, and 4MBytes of RAM. Byte, word and long-word accesses are supported. PC4b boards are equipped with a jumper to allow 28-pin or 32-pin EPROM's to be installed if desired.

Jumpers are also provided to allow the two (2 MByte) banks of EPROM to be swapped in the 68020's address space. This feature can be quite useful when debugging an EPROM based application program.

A DS1386-8 real-time clock with interrupt capability and 8 MBytes of NVRAM is provided for system use. Both RTC, and NVRAM functions are maintained by a built-in lithium battery. The NVRAM is currently used to hold module-specific information (e.g., Ethernet address). PC4b FSCC's also feature a write protect bit for the DS1386 to prevent a program from accidentally over writing the NVRAM. This bit is bit zero of the DUART's eight bit parallel I/O port, and is set and cleared by the following commands:

```
MOVE.B      #$01,$0170000E  *Write enable the NVRAM
MOVE.B      #$01,$0170000F  *Write protect the NVRAM
```

Processor reset occurs at power-up, by pressing the front-panel Reset Push-button or through the front panel reset input Lemo. A reset can also be generated by a "watchdog" timer contained in the FASTBUS Parallel Port 1 controller (68230). If enabled, this timer must be reset by software periodically. The timer can be set for any period from 4 µsec up to approximately 50 seconds. A module reset can be forced by software which drives the parallel port watchdog time-out bit low.

A Processor bus response timer is contained in the other 68230 port controller. When enabled, this timer will generate a 68020 BUS ERROR exception if the Processor fails to complete a bus cycle within the specified time. The bus response timer also serves as the FASTBUS "long timer" when operating the FPORT in CPU mode. However, since there is a one level instruction pipe-line, the time-out will occur on the FASTBUS instruction following the instruction which actually caused the FPORT controller to hang. When using the FPORT in LIST mode, the CPU is not writing instructions directly into the FPORT so the bus response timer does not time-out if the FPORT controller hangs.

2.2.1 Control and Status Registers

The FSCC has no hardware implemented control or status registers which are independent of the associated controllers. Refer to the appropriate controller description for register definitions. Since the FSCC's slave interface is software driven, any FASTBUS CSR register may be defined in software.

2.2.2 Error Responses

The FPORT Controller normally monitors the FASTBUS exception logic. When a FASTBUS error or FASTBUS Reset is detected, the controller will terminate its current operation and return to an idle state. If another FASTBUS instruction is pending (Processor pipelined mode), the FPORT Controller will return Processor DSACK immediately to clear the bus and allow interrupt processing. The instruction which was pending will not be executed. If the FPORT Controller fails to recognize an error and return control to the Processor, the Processor will eventually time-out with a BUS ERROR exception. This can happen, for example, if the address/data cycle timer is disabled and no FASTBUS acknowledge is received.

FASTBUS exceptions can be cleared by asserting the parallel port FASTBUS clear error (**FCLERR**) signal. The FASTBUS interrupt vector must be programmed and the interrupt enabled to allow recognition of FASTBUS exceptions.

Because FASTBUS operations can be queued, a FASTBUS error interrupt may not apply to the current processor data cycle. For example, a FASTBUS block transfer can be initiated and then followed by any number of non-FASTBUS processor operations while the block transfer takes place. A FASTBUS error during the block transfer generates an interrupt which may be unrelated to the current processor activity. For any FASTBUS interrupt, the parallel port status lines must be examined to determine the cause of the interrupt.

If a FASTBUS error occurs while the FPORT is operating in LIST mode, the List Sequencer tags the event as bad by setting the Bad_Event bit in the Header word of the current event. The error is then cleared and the sequencer attempts to continue executing the list.

2.2.3 Interrupts

A 68901 multifunction peripheral provides the vectored interrupt control. 16 interrupt conditions are prioritized by the 68901. Eight of these conditions are internal to the 68901, and eight are connected to the 68901's GPIP (General Purpose I/O Interrupt Port) inputs. Interrupts for the following GPIP inputs are defined on FSCC's;

Table 7 Hardware Interrupts

Interrupt Line	Name	Function
GPIP7	TFIFOOR	TFIFO Output Ready (rising edge)
GPIP6	RTCREQ*	Real-time Clock (falling edge)
GPIP5	ETHREQ	Ethernet (rising edge)
GPIP4	SERREQ*	Serial Port (falling edge)
GPIP3	FBERR*	FASTBUS error (falling edge)
GPIP2	AUXREQ*	OPORT Controller (falling edge)
GPIP1	FBREQ*	FASTBUS request (falling edge)
GPIP0	FPCREQ*	FASTBUS Port Controller (falling edge)

* Active low signal.

Interrupt vectors are programmable and interrupts can be separately enabled, disabled or masked in the 68901 controller. Refer to the Motorola 68901 manual for register definitions. The 68020 interrupt mask is hardwired to IPL2, which corresponds to a interrupt level of 4, and is not used except to enable or disable all interrupts.

TIFIFOOR-GPIP7: The Trigger FIFO's Output Ready line is connected to IRQ7. The 68901 will see an active edge on the Output Ready line and generate an interrupt, when the first Trigger Strobe clocks the FIFO, and each time an End Of Event instruction clocks a word out of the Trigger FIFO.

RTCREQ*-GPIP6: The real-time clock (Dallas Semiconductor DS1386-8) can be programmed to generate periodic interrupts at a rate of 10 msec to 100 seconds. It can also be programmed to interrupt on a specific date or/and time.

ETHREQ-GPIP5: This is the Ethernet message interrupt. Ethernet messages are buffered by the controller in Dual-Port memory so immediate interrupt response is not required.

SERREQ*-GPIP4: The serial ports can be programmed to generate interrupts when the receive buffer is loaded or the transmit buffer is empty.

FBERR*-GPIP3: FASTBUS error conditions result in a processor interrupt when enabled. The conditions are:

1)FASTBUS time-out-failure of a slave module to respond within 3 µs on an address or data cycle. This error can be disabled through the short timer enable bit (STEN) in parallel port 2.

2)FASTBUS SS errors-SS responses of 1,2,3,4,5,6 or 7 on a Address cycle or 1,3,4,5,6, or 7 on a Data cycle will cause a FASTBUS error interrupt. The last non-zero value of SS is latched at parallel port 2 (FLSS0, FLSS1, FLSS2).

3)Data FIFO overflow.

AUXREQ*-GPIP2: OPORT request to processor. The Output Port Controller Interface can be EPROM programmed to cause this interrupt on Step Acknowledge, Permit in received, or on any combination of these two events.

FBREQ*-GPIP1: (External) FASTBUS request-This interrupt is generated when the FSCC is accessed as a slave or when a FASTBUS Service Request (SR) or FASTBUS Reset (RB) is issued. RB does not directly reset the processor. A processor reset can be generated by software in the RB interrupt handler. RB does not cause an interrupt if it is being driven by the FSCC itself.

FPCREQ*-GPIP0: The FPORT controller can be programmed to assert an interrupt request at any time. In the current standard FASTBUS instruction set, the FPORT controller is programmed to generate this interrupt at the End of Block in Block and Pipelined transfers. In the current List Mode FASTBUS instruction set, an FPCREQ interrupt instruction can be inserted at any point in the list.

3. Communication Interfaces

The IEEE 802.3/Cheapernet (10BASE2) interface physical connection is through a front-panel BNC connector. The Intel 82586 coprocessor interfaces to the 68020 through a 2K by 32 bit dual-ported memory. This configuration allows both processors to operate independently and is necessary to avoid buffer overrun if the 68020 is unable to process interrupts for extended periods. It also improves the speed of FASTBUS operations since there is no contention on the processor bus.

Two standard RS-232 serial ports (Signetics 68681 DUART) are provided for development and diagnostic use. One of these will be connected to a terminal, PC, or Work Station. The other can serve as a link to a host machine. Data rates to 9600 baud will be supported. Operation of these ports is controlled by on-board software. At module initialization, the ports are configured for 9600 baud using XON / XOFF protocol. Refer to the 68681 data sheet for register definitions. The Host port connector is wired as a null-modem port.

FASTBUS is accessed through the crate segment backplane. All FASTBUS operations are performed by a microsequencer as directed by the processor. A sequence of FASTBUS transfers may be performed to the Data FIFO's or to the Processor FIFO's without processor intervention.

The Output Port is controlled by a microsequencer which is directed by the processor. A transfer may be initiated and proceed unattended. The Output Port transfers data through the Auxiliary connector of the backplane. Personality cards, which plug into the auxiliary connector from the rear of the FASTBUS crate are available for standard protocols. Specifications for available personality cards are included as appendixes to this document.

3.1 FASTBUS Interface (FPORT)

The FASTBUS Port Controller (FPORT) provides most of the low-level control of FASTBUS operations, based on simple memory-mapped instructions from the processor. It consists of two sets of three parallel EPS448 programmable sequencers plus assorted PLD's. The two sets of sequencers allows the FPORT to operate in one of two user selectable modes. Normal Mode, and List Mode. Normal or Processor mode is the default mode of operation. In Normal Mode, instructions are passed from the processor to the FPORT controller one at a time. Each instruction passed to the FPORT is individually acknowledged either after or during execution (depending on the instruction). In List Mode, the FPORT operates similarly to Normal Mode, but the FASTBUS instructions are loaded into the List FIFO memory. The List Sequencer is then enabled, and execution of the list of FASTBUS instructions is executed without processor intervention. Instructions are provided to allow self repeating lists which execute upon receipt of a Front Panel Trigger Strobe. Normal Mode allows a limited amount of FASTBUS data manipulation, and better error source determination over List Mode. List mode allows the FPORT and OPORT to work to read-out data and output it without processor intervention. List mode is usually faster due to elimination of CPU overhead.

The EPS448 is EPROM programmable and the instruction set cannot be modified by the processor. FPORT Controller output signals are defined Table 9. The EPS448 is limited to 256 states, of which only 64 support conditional branching. Therefore, the standard sequencer will directly execute only FASTBUS primitive operations. Two 68230 parallel ports provide status and control communication between the processor and the FASTBUS control logic. Parallel port lines are defined in Table 8 ("I" in the first column indicates an input signal, "O" indicates an output, "S" indicates a special function pin)

Port data direction and signal states are individually programmable. The software should avoid defining or driving INPUT pins as OUTPUTs, since multiple drives on the same signal line can cause circuit damage. If in doubt, leave ports in the normal power-on reset configuration.

Table 8 Parallel Port Bit Definitions

Parallel Port 1

Input/Output	Bit	Signal	Function	
I	A0	FRRD	FASTBUS received RD	
I	A1	FRDS	FASTBUS received DS	
I	A2	FRMS0	FASTBUS received MS0	
I	A3	FRMS1	FASTBUS received MS1	
I	A4	FRMS2	FASTBUS received MS2	
I	A5	FRAK	FASTBUS received AK	
I	A6	FRDY	FASTBUS master	
I	A7	FSLV*	FASTBUS slave mode	
I	B0	TRIG0	Trigger Vector 0	All Port B inputs are latched
I	B1	TRIG1	Trigger Vector 1	on the rising edge of the
I	B2	TRIG2	Trigger Vector 2	trigger input strobe.
I	B3	TRIG3	Trigger Vector 3	
O	B4	FP0	Front Panel 0	Front Panel differential RS-485
O	B5	FP1	Front Panel 1	Outputs
O	B6	FP2	Front Panel 2	
O	B7	FP3	Front Panel 3	
O	C0	PRS*	Processor FIFO reset	
I	C1	PEF*	Processor FIFO empty	
O	C2	COPYEN	Processor FIFO copy enable	
O,S	C3	WDTO*	"Watchdog" time-out	
O	C4	DRS*	Data FIFO reset	
O	C5	DRT*	Data FIFO retransmit	
O	C6	SRS*	Sequencer List FIFO reset	
I	C7	DFF*	Data FIFO overflow	

Parallel Port 2

Input/Output	Bit	Signal	Function
O	A0	FDGK	FASTBUS drive GK
O	A1	FDRB	FASTBUS drive RB
O	A2	FDSS0	FASTBUS drive SS0
O	A3	FDSS1	FASTBUS drive SS1
O	A4	FDSS2	FASTBUS drive SS2
I	A5	CSR/DATA*	FASTBUS MS0 latched at AS(u)
O	A6	LFIFOEN	List FIFO Enable
O	A7	LCEN	Local Counter Enable
I	B0	SSTAT0	Sequencer status 0
I	B1	SSTAT1	Sequencer status 1
I	B2	SSTAT2	Sequencer status 2
I	B3	SSTAT3	Sequencer status 3
I	B4	FRSR	FASTBUS SR
O	B5	LRT*	List FIFO Retransmit
O	B6	FCLERR	FASTBUS Clear Errors
O	B7	SNRESET*	FPORT Controller Reset
O	C0	STEN	Short timer enable
I	C1	STO*	Short time-out
I,S	C2	LTEN	Long timer enable
O,S	C3	LTO*	Long time-out
I	C4	FLSS0	FASTBUS latched SS0
I	C5	FLSS1	FASTBUS latched SS1
I	C6	FLSS2	FASTBUS latched SS2
I	C7	FRESET*	FASTBUS reset

Table 9 FPORT Sequencer Output Bits

	Bit	Name	Function
EPS448-3	F00	SMUX0	Select sequencer condition code set 0
	F01	SMUX1	Select sequencer condition code set 1
	F02	SMUX2	Select sequencer condition code set 2
	F03	FDSACK*	Processor data strobe acknowledge
	F04	SSTAT0	Sequencer status bit 0
	F05	SSTAT1	Sequencer status bit 1
	F06	SSTAT2	Sequencer status bit 2
	F07	SSTAT3	Sequencer status bit 3
	F08	CCLERR	Controller Clear Error
	F09	TIMER	Short timer enable
	F10	CEOE	Control "End-of-Event"
	F11	DEOE	Data "End-of-Event"
	F12	FCLK	Global word counter clock
	F13	STATCLK	Sequencer status clock
	F14	SR*	Sequencer List FIFO read
	F15	PFIFOEN	Processor FIFO input enable
EPS448-2	F00	FSAS	FASTBUS set AS
	F01	FCAS	FASTBUS clear AS
	F02	FSDS	FASTBUS set DS
	F03	FCDS	FASTBUS clear DS
	F04	FSDK	FASTBUS set DK
	F05	FCDK	FASTBUS clear DK
	F06	FDWT	FASTBUS drive WT
	F07	FDMS0	FASTBUS drive MS0
	F08	FDMS1	FASTBUS drive MS1
	F09	FDMS2	FASTBUS drive MS2
	F10	FDRD	FASTBUS drive RD
	F11	FDEG	FASTBUS drive EG
	F12	MUX3*	List Instruction Select
	F13	FEOBA	FASTBUS EOB Acknowledge
	F14	FREQ	FASTBUS request bus
	F15	FREL	FASTBUS release bus
EPS448-1	F00	DFIFOEN	Data FIFO input enable
	F01	FPCREQ*	Sequencer Interrupt Request (TRAP)
	F02	FCOE*	FASTBUS control output enable
	F03	FDOE*	FASTBUS data output enable
	F04	SDG*	Data pipeline latch enable
	F05	DSBA	Data pipeline latch B->A mode control
	F06	DCPBA	Data pipeline latch B->A clock
	F07	DSAB	Data pipeline latch A->B mode control
	F08	SDCPAB	Data pipeline latch A->B clock
	F09	DDIR	Data pipeline latch direction
	F10	SPOE*	Processor transceiver output enable
	F11	SDW	Data FIFO write
	F12	SRT*	Sequencer List FIFO retransmit
	F13	SLCOE	Local word counter output enable
	F14	LC0	Local word counter control 0
	F15	LC1	Local word counter control 1

Table 10 shows a list of supported memory mapped FASTBUS master operations permitted by the standard FPORT sequencer instruction set.

Table 10 Normal Mode FASTBUS Instructions

FPORT Base addresses for PC4a FSCC's:

SLOWBASE	EQU	$01C00000
FASTBASE	EQU	$01B80000
USRBASE	EQU	$01E80000

Address (HEX)	FASTBUS Operation	Status Code
SLOWBASE+300	BUS_ARBITRATE	1
FASTBASE+004	BUS_RELEASE	2
FASTBASE+304	ADDRESS_DATA_GEOGRAPHICAL	3
FASTBASE+308	ADDRESS_CSR_GEOGRAPHICAL	3
FASTBASE+30C	ADDRESS_DATA_LOGICAL	3
FASTBASE+310	ADDRESS_CSR_LOGICAL	3
FASTBASE+314	ADDRESS_DATA_BROADCAST	3
FASTBASE+318	ADDRESS_CSR_BROADCAST	3
FASTBASE+31C	ADDRESS_RELEASE	4
SLOWBASE+320	DATA_PROCESSOR_RANDOM_READ	5
FASTBASE+324	DATA_PROCESSOR_RANDOM_WRITE	5
SLOWBASE+328	DATA_PROCESSOR_SEC_ADDRESS_READ	6
FASTBASE+32C	DATA_PROCESSOR_SEC_ADDRESS_WRITE	6
SLOWBASE+008	DATA_PROCESSOR_BLOCK_TRANSFER_READ	5
FASTBASE+00C	DATA_PROCESSOR_BLOCK_TRANSFER_WRITE	5
FASTBASE+330	DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE	5
FASTBASE+334	DATA_FIFO_BLOCK_TRANSFER_READ	7
FASTBASE+338	DATA_FIFO_PIPELINED_READ_100	8
FASTBASE+33C	DATA_FIFO_PIPELINED_READ_200	8
FASTBASE+340	DATA_FIFO_PIPELINED_READ_400	8
FASTBASE+020	SEQUENCER_NULL	C
SLOWBASE+028	BULB_TEST	B
SLOWBASE+010	LOCAL_COUNTER_LOAD	9
SLOWBASE+014	LOCAL_COUNTER_READ	9
SLOWBASE+018	FIFO_WRITE_DATA	A
SLOWBASE+024	END_OF_EVENT	E
SLOWBASE+02C	END_OF_EVENT_REXMIT	E

Table 11 gives a list of supported slave instructions. Slave instructions are not supported in List Mode.

Table 11 Normal Mode FASTBUS Slave Instructions

SLOWBASE+01C	SLAVE_DATA_INPUT	D
FASTBASE+344	SLAVE_DATA_OUTPUT	D

The following instruction has an effect only if executed after sequencer reset and before any other FASTBUS instruction is executed. If executed any other time, it has the same effect as a NULL instruction.

USRBASE+020	USER_SEQUENCER_SELECT	C

Refer to Appendix A for detailed instruction definitions.

3.1.1 FASTBUS Controller Operation

Since the 68020 is an asynchronous processor, when it initiates a processor bus cycle it must wait for an acknowledge signal before continuing with the next bus cycle. The FASTBUS instructions Table 10 are seen by the processor as functioning in one of two ways. "Fast" instructions return the processor data acknowledge signal immediately upon being invoked (a memory mapped instruction is invoked by addressing it), whereby "Slow" instructions do not return the processor data acknowledge until later in the microsequencer routine. The 68020 is held in the middle of a bus cycle, unable to execute any other instructions until the acknowledge signal becomes true. Therefore, slow FASTBUS instructions are used only where necessary. An example is the DATA_PROCESSOR_RANDOM_READ instruction. This instruction does not return the acknowledge signal until it has received and latched the data received from the FASTBUS slave. The 68020 sees the instruction similarly to a simple 32-bit memory read. If the FASTBUS short timer is enabled and the slave does not respond, the time-out would cause the processor data acknowledge to be returned, the current FASTBUS instruction to be aborted, and a 68020 interrupt. If the short timer is not enabled, the long timer would eventually time out, and cause a 68020 bus error exception. The FASTBUS sequencer would then have to be reset since it would still be waiting for the slave to respond. In general, instructions which return data immediately to the 68020 are slow instructions. All other instructions are fast instructions. Slow instructions can be identified by their base address of 01C00000, fast instructions have a base address of 01B80000.

3.1.2 FASTBUS Arbitration

The FSCC supports standard and assured access arbitration. It does not support prioritized arbitration. Bits 0-5 of the processor data bus contain the arbitration vector. Bit 7 determines whether the assured access protocol is active. Bit 6 and 8-31 are ignored. These bit assignments correspond to CSR 8. A FASTBUS arbitration can be performed by a processor instruction of the form -

```
MOVE.L      CSR8 , BUS_ARBITRATE
```

Since BUS_ARBITRATE is a slow instruction, the processor does not receive its acknowledge until the FASTBUS arbitration cycle has been won. If the arbitration cycle is not won before the long timer times out, a bus error will occur. The bus can also be acquired simply by asserting the GK line through the parallel port. Note that this is not a standard FASTBUS operation and is provided only for single master systems without ancillary logic.

3.1.3 FASTBUS Reset Bus (RB)

The Processor can issue a FASTBUS RB (Reset Bus) signal directly through the parallel port. In this case the parallel port GK signal should be asserted simultaneously. RB does not cause a processor interrupt when it is driven by the FSCC itself.

3.1.4 FASTBUS Slave Mode Operation

The FSCC supports a limited slave mode through processor emulation. Geographical address recognition logic is contained in hardware. All other slave functions are controlled by the processor. FASTBUS WT is asserted on each DS transition. The data cycle response time is dependent on software and is typically 10-15 µsec per FASTBUS word.

Accessing the FSCC as a slave causes a Processor interrupt. The processor must poll the DS, MS, RD, AK, and CSR/DATA* lines via parallel port inputs. The SS response is then placed on the bus via parallel port outputs and a slave mode input or output operation is executed by the sequencer. The CSR/DATA* line reflects the status of MS0 which has been latched during the address cycle.

The two FASTBUS Slave instructions cause a single word of data to be transferred between the processor and FASTBUS with an associated DK transition. WT is released prior to the DK transition and is reasserted on the next DS transition. Any FASTBUS CSR or DATA location can be defined by processor software.

The FSCC is limited in its ability to execute master and slave operations simultaneously, since the processor is involved in both cases. There is normally no need for the FSCC to address itself, with the possible exception of crate mapping. If the FSCC does address itself, the slave logic will attach (return AK) but no data cycles will be possible. A slave mode interrupt is generated and the processor software must then recognize the simultaneous master/slave condition by examining the parallel port **FSLV*** bit. The software should bypass any FASTBUS data cycles and retrieve the information directly from internal memory.

3.1.5 List Mode FASTBUS Operation

LIST Mode FASTBUS operation was enhanced on PC4 boards (carried over to PC4a/b modules) by adding 8 bits to the 32-bit wide List FIFO to allow instructions to be stored in the list as well as data operands. This allows the LIST FIFO to be used similarly to PC3 versions with the added flexibility of being able to change the instruction list without developing special application-specific microcode. All PC4b FSCC's are assembled with generic LIST FIFO driver microcode installed in the user microsequencer sockets on the board. This microcode works similarly to the standard microcode except that, when enabled, it takes its FASTBUS instructions and data from the LIST FIFO instead of the 68020 directly.

The LIST FIFO is controlled by three parallel port bits. The SRS* bit is a low true reset bit for the LIST FIFO. To reset the LIST FIFO the SRS* bit must be pulsed low with two processor writes to parallel port 1 bit C6. The second LIST FIFO control bit in the parallel port is the LFIFOEN bit. LFIFOEN is a high true signal which enables the data path between the LIST FIFO and the FPORT controller (Parallel Port 2 bit A6). The third parallel port bit controlling the LIST FIFO is the LRT* bit (Parallel Port 2 bit B5). This bit is the low true retransmit bit to the LIST FIFO chips. When pulsed low, it causes a previously executed list to be executed again.

The FASTBUS master operations listed in Table 10 are permitted by the PC4b LIST FIFO sequencer instruction set. The List Mode FASTBUS instruction set description is included as an appendix to this document.

Table 12 List Mode FASTBUS Instructions

FPORT List FIFO Base addresses for PC4b FSCC's:

LISTBASE	EQU	$01E00000
USRBASE	EQU	$01E80000

Address(HEX)	FASTBUS Operation	Status Code
LISTBASE+300	BUS_ARBITRATE	1
LISTBASE+004	BUS_RELEASE	2
LISTBASE+304	ADDRESS_DATA_GEOGRAPHICAL	3
LISTBASE+308	ADDRESS_CSR_GEOGRAPHICAL	3
LISTBASE+30C	ADDRESS_DATA_LOGICAL	3
LISTBASE+310	ADDRESS_CSR_LOGICAL	3
LISTBASE+314	ADDRESS_DATA_BROADCAST	3
LISTBASE+318	ADDRESS_CSR_BROADCAST	3
LISTBASE+31C	ADDRESS_RELEASE	4
LISTBASE+320	DATA_RANDOM_READ	5
LISTBASE+324	DATA _RANDOM_WRITE	5
LISTBASE+328	DATA _SECONDARY_ADDRESS_READ	6
LISTBASE+32C	DATA _SECONDARY_ADDRESS_WRITE	6
LISTBASE+00C	DATA _BLOCK_TRANSFER_WRITE	5
LISTBASE+330	DATA _BLOCK_TRANSFER_TERMINATE	5
LISTBASE+334	DATA _BLOCK_TRANSFER_READ	7
LISTBASE+338	DATA _PIPELINED_READ_100	8
LISTBASE+33C	DATA _PIPELINED_READ_200	8
LISTBASE+340	DATA _PIPELINED_READ_400	8
LISTBASE+344	DATA_RANDOM_READ_TO_LOCAL_COUNTER	5
LISTBASE+348	DATA _BLOCK_READ_LEADING_WORD_COUNT	7
LISTBASE+02C	TRIGGER_HOLD	C
LISTBASE+044	TRIGGER_HOLD_WITH_HALT_REQUEST	C
LISTBASE+030	INSTRUCTION_LIST_RE-EXECUTE	B
LISTBASE+01C	GENERATE_FPCREQ (IRQ)	A
LISTBASE+034	POLL_HALT_REQUEST	0
LISTBASE+038	DELAY2 (2 microseconds)	B
LISTBASE+03C	DELAY10 (10 microseconds)	B
LISTBASE+040	DELAY100 (100 microseconds)	B
LISTBASE+020	SEQUENCER_NULL	C
LISTBASE+028	BULB_TEST	B
LISTBASE+010	LOCAL_COUNTER_LOAD	9
LISTBASE+014	LOCAL_COUNTER_READ	9
LISTBASE+018	FIFO_WRITE_DATA	A
LISTBASE+024	END_OF_EVENT	E
	FB_ERROR	F

The following instruction has an effect only if executed after sequencer reset and before any other instruction is executed. If executed any other time, it has the same effect as a NULL instruction.

USRBASE+020	USER_SEQUENCER_SELECT	C

Initializing the LIST FIFO is done in four steps:

1) Reset the LIST FIFO by pulsing the SRS* bit. (Recommended but not necessary if FIFO is known to be empty).
2) Load an instruction stream into the List FIFO using the instructions in Table 12 (LFIFOEN should be false while List FIFO is loaded).
3) Enable the LIST microsequencer by accessing the address: FPORTUSR+020. (Only needs to be done once after reset, must be done before any FASTBUS instructions are executed and must be done while LFIFOEN is false. Has no effect if executed more than once).
4) Enable the LIST FIFO by setting the LFIFOEN bit true. The FPORT controller will then start executing the list.

* Sample crate readout program using the LIST FIFO:

```
*  PP1CDATA is parallel port 1C's data register
*  Reset LIST FIFO.
   ANDI.B      #$BF,PP1CDATA                  *Set LIST FIFO Reset bit true
   ORI.B       #$40,PP1CDATA                  *Clear LIST FIFO Reset bit.

*  Load LIST FIFO with FASTBUS instruction stream.

   MOVE.L      #0,LIST_POLL_HALT_REQUEST      *test for halt req. from CPU
   MOVE.L      #0,LIST_TRIGGER_STROBE_HOLD    *wait for a trigger
   MOVE.L      #0,LIST_DELAY2                 *delay 6 microseconds
   MOVE.L      #0,LIST_DELAY2
   MOVE.L      #0,LIST_DELAY2

   MOVE.L      CSR8,LIST_BUS_ARBITRATE        *Arbitrate

   MOVE.L      PRIMADD1,LIST_ADD_DATA_GEO     *Address first slave
   MOVE.L      SECADD1,LIST_SEC_ADD_WR        *Secondary address
   MOVE.L      WORDCNT1,LOCAL_COUNT_LOAD      *Load Local Word Counter
   MOVE.L      #0,LIST_FIFO_BLOCK_READ        *Block read to data FIFO
   MOVE.L      #0,LIST_ADDRESS_RELEASE        *Address Release

   MOVE.L      PRIMADD2,LIST_ADD_DATA_GEO     *Address second slave
   MOVE.L      SECADD2,LIST_SEC_ADD_WR        *Secondary address
   MOVE.L      WORDCNT2,LOCAL_COUNT_LOAD      *Load Local Word Counter
   MOVE.L      #0,LIST_FIFO_BLOCK_READ        *Block read to data FIFO
   MOVE.L      #0,LIST_ADDRESS_RELEASE        *Address Release

   MOVE.L      PRIMADD3,LIST_ADD_DATA_GEO     *Address third slave
   MOVE.L      SECADD3,LIST_SEC_ADD_WR        *Secondary address
   MOVE.L      WORDCNT3,LOCAL_COUNT_LOAD      *Load Local Word Counter
   MOVE.L      #0,LIST_FIFO_BLOCK_READ        *Block read to data FIFO
   MOVE.L      #0,LIST_ADDRESS_RELEASE        *Address Release

   MOVE.L      #0,LIST_BUS_RELEASE            *Bus Release

   MOVE.L      #0,INSTRUCTION_LIST_RE-EXECUTE *repeat list

*  Enable User microsequencer (switch FPORT controller from standard microcode
```

```
*       to LIST microcode).
        ANDI.B      #$7F,PP2BDATA                   *reset FPORT controller
        ORI.B       #$80,PP2BDATA

        MOVE.L      #0,USR_SEQUENCER_SELECT         *FPORT User Sequencer Select

*       Enable List FIFO (List execution will start after this instruction is executed)
        ORI.B   #$40,PP2ADATA                       *Set PP2A bit 6 high (LFIFOEN)
```

The List Sequencer can continuously execute the same list repeatedly. This is done by putting an INSTRUCTION_LIST_RE-EXECUTE instruction at the end of the FASTBUS instruction list in the List FIFO. When the List Sequencer sees the re-execute instruction, it toggles the RETRANSMIT input to the List FIFO. This causes the FIFO to reset its internal pointers, and list execution begins at the first list instruction.

To execute a graceful halt of the List Sequencer while operating in a self re-executing list, a POLL_HALT_REQUEST instruction may be inserted at any convenient point in the list. When executed, the List Sequencer checks for an FPORT List Halt Request from the processor. An FPORT List Halt Request instruction is a memory mapped CPU instruction. It is executing by reading this memory location (See Table 6). Execution of a Halt_Request CPU instruction sets a status flag that the List Sequencer can test. The processor is then paused in the middle of a cycle, waiting for an acknowledge from the List Sequencer. When the List Sequencer executes the POLL_HALT_REQUEST instruction, the status flag is tested. If it is true, the sequencer returns the processor acknowledge and halts. If the flag is not true, list execution continues at the next instruction. If the POLL_HALT_REQUEST instruction is not executed within the Long Timer timeout period, then a 68020 Bus Error Exception will result. If a second FPORT List Halt Request instruction is executed after the List Sequencer has already been halted, it does not return an acknowledge and a Bus Error Exception occurs. A Bus Error Exception will also result if a Halt_Request CPU instruction is executed and the FASTBUS Sequencer is not in List mode.

3.1.6 Data Transfer Description and Transfer Rates

The FSCC supports FASTBUS Pipelined Transfer rates of 100, 200 and 400 nsec per word. At the 100 nsec per word rate, End-of-Block is not normally returned in time to avoid another DS transition. Therefore, Pipelined transfers at 100 nsec may read one word beyond the end of block and this word will be included in the output data stream.

The FSCC also supports Block Transfers at a rate of ~150 nsec per word. The actual transfer rate will be 100 nsec plus the slave DS-DK response time, rounded to the next highest 50 nsec increment.

Block and Pipelined Transfers directed to the Data FIFO are functional for Read mode only. Data is placed in the FIFO and is not accessible to the Processor. However, data can be simultaneously routed to the Processor FIFO by setting the COPYEN bit in the parallel port. A Block Transfer Read/Write instruction for Processor memory is available, but operates at the same speed as single word Processor read and write operations. It implements one step of the block transfer for each processor MOVE instruction executed. The processor memory block transfer must be terminated with a DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE instruction.

Although the Data FIFO is limited to blocks of 4094 on PC4b boards, larger blocks can still be transferred provided that the destination is capable of interpreting a word count of 4094 as a partial transfer. For block transfers which are some exact multiple of 4094 words, a final block of length zero must be transferred.

3.1.7 Internal Control and Status Registers

All Control and Status Registers related to FASTBUS operation are implemented in 68020 software. The number of registers is limited only by available processor memory.

3.1.8 Error Responses

The FPORT Controller latches FASTBUS error conditions but does not attempt any recovery or retry. A processor interrupt is generated and the processor has the option of attempting recovery or, more likely, skipping the entire readout sequence. Because of internal pipelining, errors may not be reported immediately with the bus cycle generating the error. Also, the amount of status information available to the processor is limited.

The standard FASTBUS instructions will abort on errors and return to the processor with an interrupt. The Standard microcode currently sets an error flag in the header word of the current event to indicate that the FASTBUS event readout failed and the data should be discarded. Note that an End-of-Event instruction must be executed before the FASTBUS error is cleared in order for the bad event to be output with the bad event flag set. If outputting the bad event is not desired, the data FIFO's and the Output Port Controller may be reset to flush the event. The FPORT Controller does not have the ability to test status flags in slave modules to locate problems, so error recovery through microcode is generally limited to reset and continue type operations.

If a FASTBUS error occurs during a readout using the Instruction List FIFO, the bad event flag is set in the header word (word count word), and the data is immediately output. Any further readout of this event is not attempted. The List microcode then clears the FASTBUS error flag, and continues on with the next list instruction. The bad event flag is bit 17 of the first word (word count word) of the data stream.

3.2 Ethernet

The INTEL 82586 LAN coprocessor performs message framing management in transmission and reception functions. It acts as a bus master, accesses memory by DMA, carries out message error checking, collision recovery functions, etc. The INTEL 82C501 Ethernet Serial Interface (ESI) implements Manchester encoding/decoding and clock recovery. ESI functionality may be checked by the processor using Loopback mode. The National DP8392 Ethernet Transceiver performs collision detection and interfaces to the coaxial cable.

3.2.1 Ethernet Interface

The INTEL 82586 coprocessor interfaces indirectly to the 68020 CPU system through a 4K by 16 bit dual ported memory. Figure 4 is a Block Diagram of the Ethernet Interface.

The dual port RAM speeds up both systems as the processor does not release the bus each time a DMA occurs, and controller bus latency is reduced to zero.

Direct interface is provided with CA (Channel Attention) and IRQ (Interrupt Request) lines. The 68020 CPU drives CA to get the attention of the controller, indicating that new commands were included in the command list to be processed.

The 82586 Ethernet controller uses IRQ to interrupt the processor when a command is complete or upon message reception.

Loop-Back mode through the 82C501 can be enabled by writing to the Loop-Back enable register, and disabled by writing to the Loop-Back disable register (see Table 6). Loop-Back through the 82C501 is also disabled by Reset.

The shared DPRAM structure is composed of four parts: The Initialization Root, the System Control Block (SCB), the Command List and the Receive Frame Area (RFA). The Initialization Root is fixed in memory. The Ethernet controller addresses that variable as **$FFFFF6**, but in fact, due to the partial decoding, its physical location is **DPRAMBASE+$1FF6**. The base address of the DPRAM from the CPU side is listed in Table 6, and is **$0000** from the Ethernet controller. Transmission and reception messages are split into small buffers to better use the available memory. The buffers, when necessary, are chained in frames. Transmission and reception buffer descriptors are accessible through the SCB table pointers.

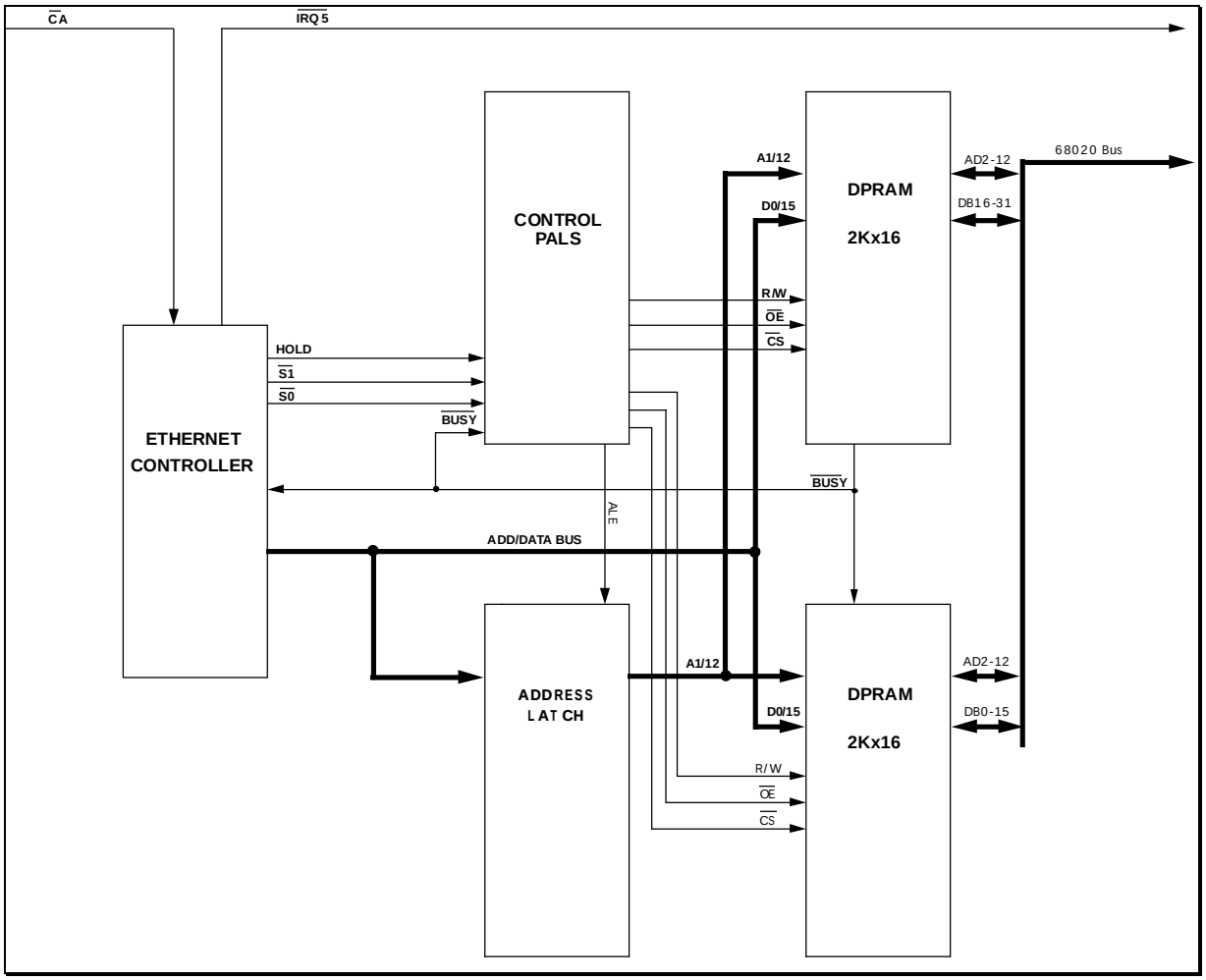


Figure 4 Ethernet Interface Block Diagram

3.2.2 Ethernet Controller Interface

The interface between the Ethernet controller and the dual ported RAM uses a 16-bit address latch implemented in 74Fxx series logic to demultiplex the Address and Data lines, and a state machine implemented in two 22V10 PALs to control the memories and the latch. The state diagram is shown in Figure 5. The PC4/a/b interface and the PC3 interface are functionally identical from both the point of view of the 68020 and of the 82586, so software for the two interfaces is the same.

3.2.3 Timing Diagrams

The Ethernet controller was designed to share an external bus, it asserts a HOLD line to keep the bus when performing burst memory cycles. In this implementation the controller has a private bus to the DPRAM, so hold acknowledge HLDA is returned instantaneously, and the state machine goes directly to IDLE to wait for the beginning of a cycle.

S1 and S0, from the Ethernet controller indicate the type of cycle the 82586 is attempting: S1=0, S0=1 is a read cycle and S1=1, S0=0 is a write cycle. Those lines are driven only during T1 and T2. The interface state machine has a separate path for write and read operations.

Figure 6 and Figure 7 show timing diagrams for 82586 read and write cycles.

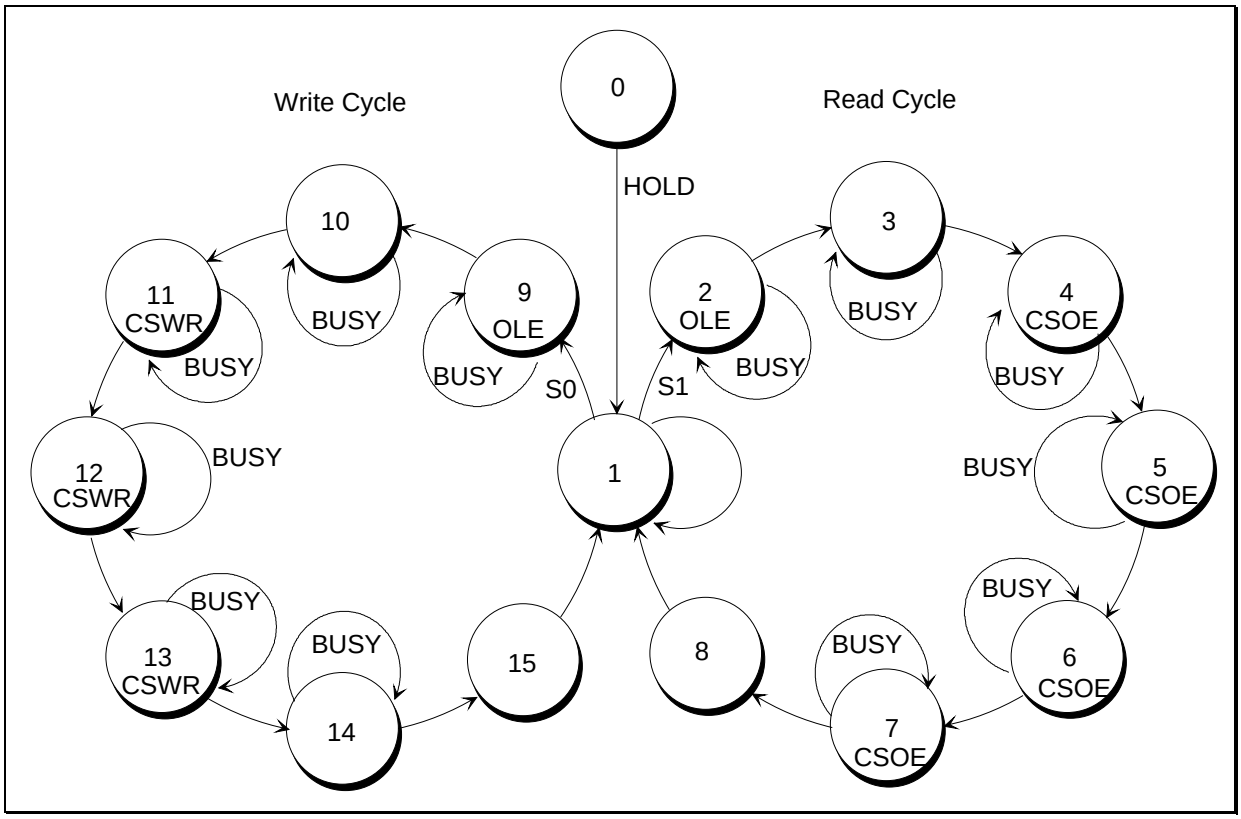


Figure 5 Ethernet Controller State Diagram

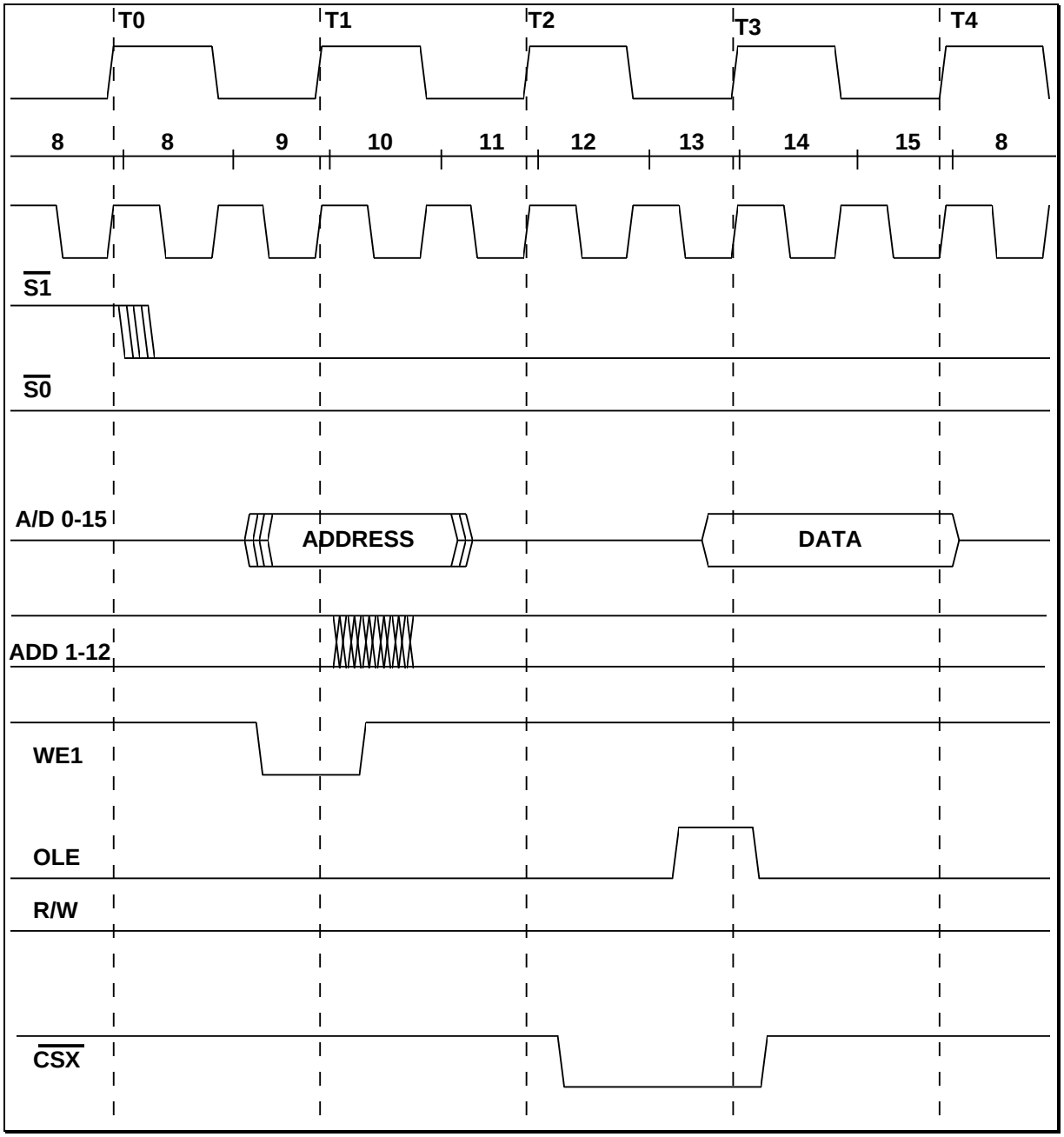


Figure 6 Ethernet Controller Read Cycle

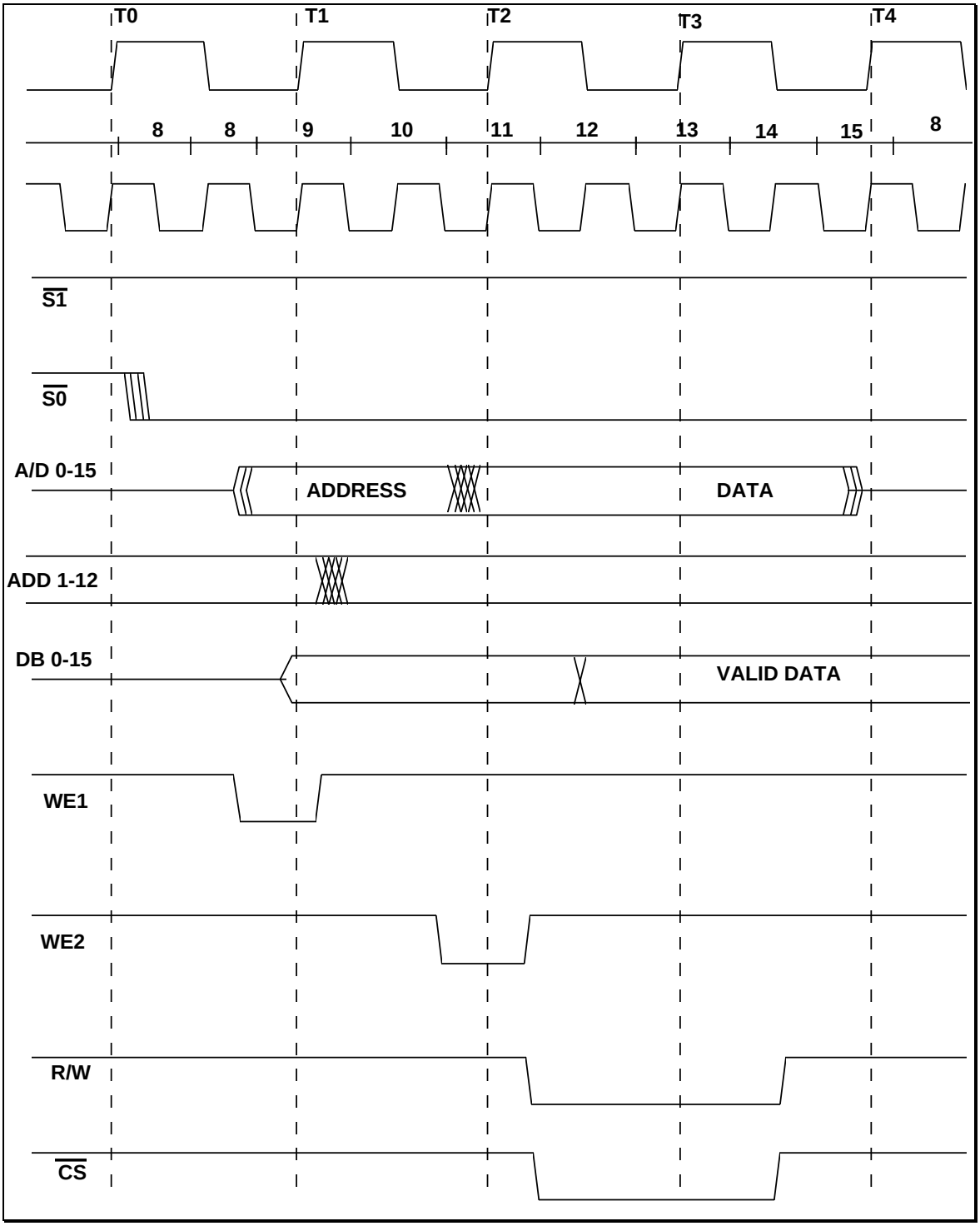


Figure 7 Ethernet Controller Write Cycle

3.3 Description and Operation of OUTPUT PORT (OPORT)

The basic function of the Output Port Controller (OPORT) is to provide a continuous stream of data with strobes for transmission via RS-485 buffers on the auxiliary card. A Block diagram of the Output Port is shown in Figure 8. For this application, it need only multiplex the Control FIFO and Data FIFO outputs and generate appropriate timing. The control FIFO contains the H&C Word (Word Count and Header bits), and the data FIFO contains the event data stream. When End-Of-Event is received from the FASTBUS controller, the OPORT controller pulls one word out of the control FIFO and outputs it. It then pulls data words out of the data FIFO until the Data End-Of-Event word (Event Delineation flag) is seen at the Data FIFO output. Connection to more complicated external bus formats will require reprogramming of the OPORT Controller or addition of a more sophisticated sequencer on the auxiliary card.

All levels at the output port are single ended TTL. Level adapters to different protocols will be mounted on the FASTBUS auxiliary card. The data path is 32 bits wide. Two control lines are provided to regulate data flow: the WAIT input pauses the OPORT and STROBE is a synchronous data strobe output.

In some applications the auxiliary bus (OPORT data cable) will be common to several FSCC modules. The PERMIT_IN and PERMIT_OUT front panel connections provide a "token passing" mechanism for enabling and disabling data output. The first FSCC in the token chain is designated “first” by setting the appropriate OPORT mode (see Table 13). Permit_Out of the “first” FSCC is connected to Permit_In of the next FSCC in the chain. Permit_Out of this module is connected to Permit_In of the next, and so on. Permit_Out of the last module in the chain is connected to Permit_In of the first module. Each FSCC in the permit chain must have its OPORT configured in the appropriate mode for its position in the permit chain (first, middle, or last).

To use a single FSCC by itself on a data cable, the OPORT is configured to be “Only” (see Table 13).

The OPORT Controller performs arbitration and control for the Data and Control (Word Count/Header) FIFO outputs. The operating mode is selected by the processor. PC4a/b FSCC’s are equipped with an OPORT controller which has a user programmable output rate. Rates of 10 MHz, 6.67 MHz, or 5.0 MHz can be software selected (Table 13).

3.3.1 OPORT Controller Operating Modes

Bits 2-0 in the OPORT Control register Select one of eight possible OPORT operating modes.

Table 13 OPORT Controller Operating Modes

Mode	Hex-Code	Function
Disabled	$00	OPORT is off-line. Outputs are tri-stated.
Token_Middle	$01	OPORT outputs a header and the event data upon receipt of a PERMIT_IN signal. This mode is usually used for a **middle FSCC in a token passing chain.** PERMIT_IN must be received before data is output, and PERMIT_OUT is generated after outputting an entire event. The EOR output driver is disabled in this mode.
Token_First	$02	OPORT outputs a header and the event data with no PERMIT_IN required for the first event after configuration. Subsequent events require a PERMIT_IN before output begins. This mode is usually used for the **first FSCC in a token passing chain.** The EOR output driver is disabled in this mode. PERMIT_OUT is generated after outputting an entire event.
Token_Only	$03	OPORT outputs a header and the event data with no PERMIT_IN required. This mode is used when there is **only one FSCC on a data cable**, and token passing is not needed. The EOR output driver is enabled in this mode. EOR is driven true for ~150ns after each event has been output. PERMIT_OUT is generated after outputting an entire event.
Rate Select	$04	Selecting and then deselecting (set to mode 0) this mode causes the OPORT controller to output data at a 6.67 MHz rate. Selecting and then deselecting this mode twice causes the OPORT controller to output data at a 5.0 MHz rate. Output rate is reset to 10 MHz by resetting the OPORT controller. The data rate set can be checked by reading the OPORT status code when the OPORT is in RESET state. OPORT status codes are listed in Table 15.
reserved	$05	Same as mode 0.
reserved	$06	Same as mode 0.
Token_Last	$07	This mode is identical to Token_Middle mode except that the OPORT controller drives the End_Of_Record signal true for ~150ns after the last data word has been output, and before the PERMIT_OUT pulse is sent. This mode is usually used for the **last FSCC in a token passing chain.** The EOR output driver is enabled whenever Token_Last mode is set.

Token_Middle: This is the normal data taking mode for a board in the **middle** (neither first nor last) **of the token passing chain**. Data is transmitted to the output daughter card with synchronous, no handshake protocol at the selected data rate. The data contains a non-inclusive leading word count, (the lower order 12 bits). Bits 12-16 of this word contain a processor programmable header, usually the Trigger ID, and bit 17 is an error flag. When the error flag is high, this indicates that the FSCC encountered a FASTBUS error during the read out process, and the data may be bad or incomplete. Bits 18-31 of the control word are undefined. When in Token_Middle mode, the OPORT controller waits until the PERMIT_IN pulse has been received. Then the controller waits for a control word (Header/Word Count) to be loaded into the control FIFO. After these two conditions have been met, data transmission begins. The header word from the control FIFO (CFIFO) is transmitted first, followed by the specified number of words from the data FIFO. Transmission ends when the End-Of-Event data word is encountered. This word is inserted into the data FIFO by the FASTBUS controller when an End-Of-Event instruction is executed (See FPORT Controller Instruction Set). The OPORT Controller then outputs a PERMIT OUT pulse, and disabled its outputs. Wait states can be inserted to slow data transmission by software commands to the OPORT controller. Three different data rates are implemented, the default rate is 10 MHz which provides 40 MBytes per second. The other two user selectable rates are 6.67 MHz, and 5.0 MHz, providing 26.68 MBytes per second, and 20 MBytes per second respectively. Either of the two optional data rates can be selected by

toggling the OPORT controller into *Rate Select Mode* (see Rate Select Mode). Refer to Figure 10 for an example of Output Port timing in Token_Middle mode.

***Token_First*:** This mode is identical in function to Token_Middle with the exception that the OPORT controller does not wait for a PERMIT_IN pulse before outputting its first event after configuration. A PERMIT_IN token must be received before any subsequent events can be output. This mode is generally used for the **first FSCC in the token passing chain.** Typically, the Permit Out output of the last board in a token passing chain, will be connected to the Permit In of the first board in the chain. The first board in the chain will be set in Token_First mode. After system reset and configuration (assuming appropriate readout code is running), the first readout trigger will cause the first board in the chain to readout its crate, output its event, and pass the token. The first board will then wait until it receives the token from the last board, before outputting another event.

Token_Last: This mode is intended for use when the FSCC is placed at the **end of the token passing chain**. This mode is functionally equivalent to Token_Middle, except that upon completion of the data transfer, the OPORT drives the End Of Record output true for 150 ns, before outputting the Permit Out pulse. The EOR output driver is enabled whenever Token_Last mode is selected.

Token_Only: This mode is intended to be used when the FSCC is the **only data source on a data cable**. In Token_Only mode, the OPORT is operating as "First" and "Last" in the permit chain. Actually, PERMIT_IN is ignored. Data is always output as soon as an End_Of_Event instruction is executed. End_Of_Record is driven in a similar manor to Token_Last mode.

Rate Select Mode: The default output rate of the OPORT is 10 MHz. Each time Rate Select Mode is selected, the OPORT reduces its output rate by one step. First from 10 MHz to 6.67 MHz, then from 6.67 MHz to 5.0 MHz. To set the OPORT into 6.67 MHz mode, Rate Select Mode is selected. The OPORT is then set in the desired output mode. All subsequent data transfers out of the OPORT will then be at a 6.67 MHz rate until the OPORT is reset. To reduce the rate to 5.0 MHz, Rate Select Mode must be selected twice. To do this, Rate Select Mode is first selected. Mode 0 is then selected, and Rate Select Mode is selected again. One of the output modes may now be selected. Data transfers will now be at 5.0 MHz until the OPORT is reset.

Refer to Figure 9 for a state transition diagram of the OPORT Controller. At power up or RESET, the state machine is in RESET state. A Rate Select command will cause the next lower speed state machine to be selected. The 10 MHz, 6.67 MHz, and 5.0 MHz state machines function identically with the exception of either one or two wait states inserted into the data outputting loop. Receipt of a Token_First or Token_Only mode selection, takes the controller directly to IDLE, while a Token_Middle or Token_Last mode selection puts the controller into PINWAIT. A PERMIT_IN will then take the controller from PINWAIT to IDLE. As soon as the control FIFO has a word in it, the controller will move from IDLE to SETUP state. In SETUP state the control FIFO is read, and the Mode Group select (MG_SEL) output is driven. The MG_SEL output going true, causes the OPORT's CPU interface to provide a second set of Mode inputs to the OPORT controller. The OPORT controller then looks at the second mode group inputs to see if the Suppress Zero Event mode is selected. If SZE mode is selected, this means that the SZE bit is set, and the next event in the Data FIFO has no data in it (a Null event). The OPORT controller then discards the control word, and proceeds to CHKMODE state. If SZE mode is not true, the controller continues on to LATCH1 state where the control word is latched, and then to STROBE state where it is strobed out of the port. The state machine then toggles between STROBE, and LATCH2 clocking data out of the data FIFO, and strobing it out the OPORT. LATCH2 state watches for the Event Delineating flag coming out of the Data FIFO, and breaks out of the loop when it becomes true. CHKMODE checks to see which mode is set to determine how to terminate the OPORT operation. If Token_Last mode is selected, DEOR state is executed which drives the End Of Record output true for 150ns. All other modes cause the state machine to go to PERMOUT state which drives the Permit Out line true for 300 ns, and then disable its output drivers. The state machine then proceeds to PERMIN2 state and waits for either a PERMIT_IN token, or a mode change to occur.

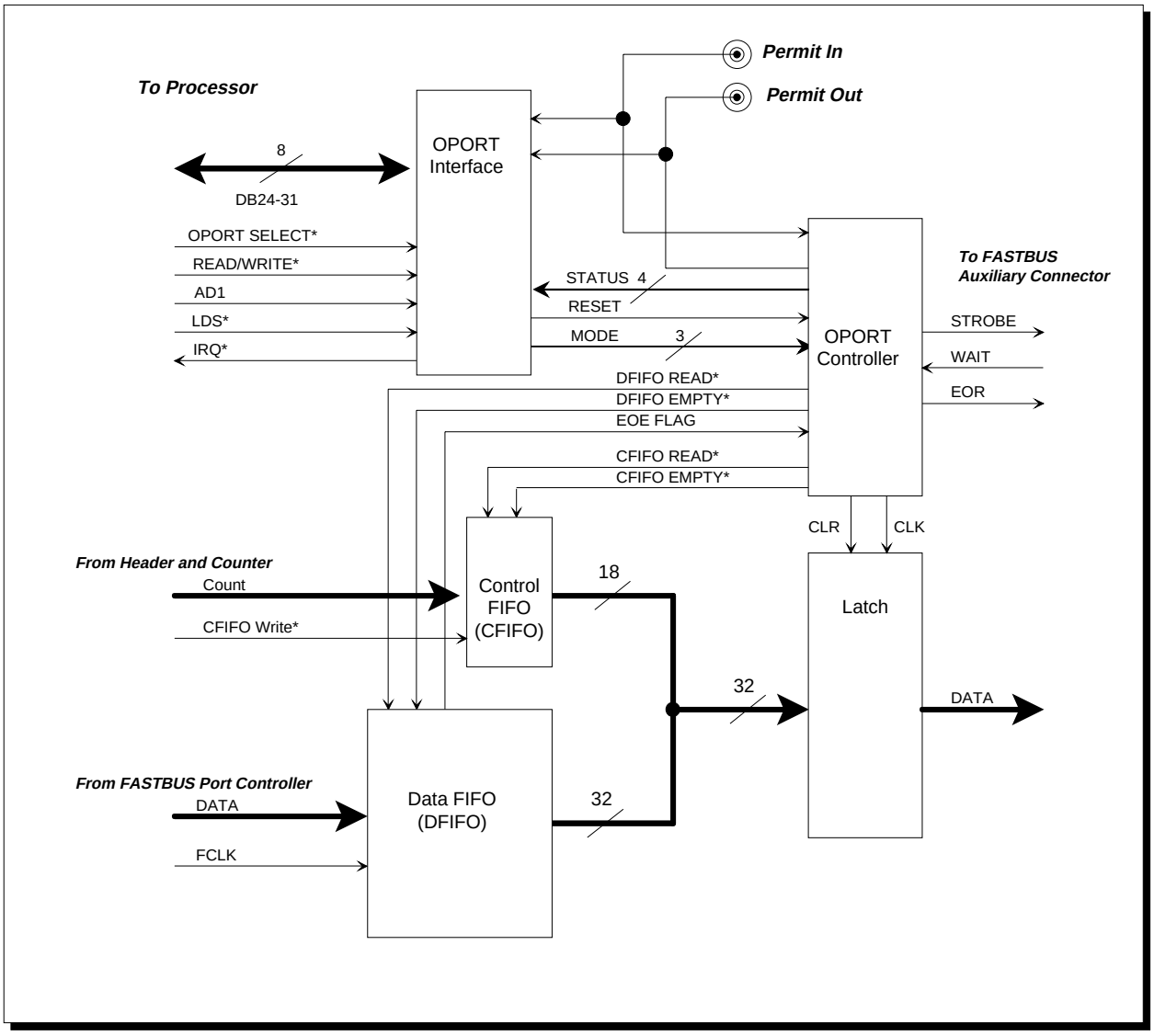


Figure 8 Output Port (OPORT) Block Diagram

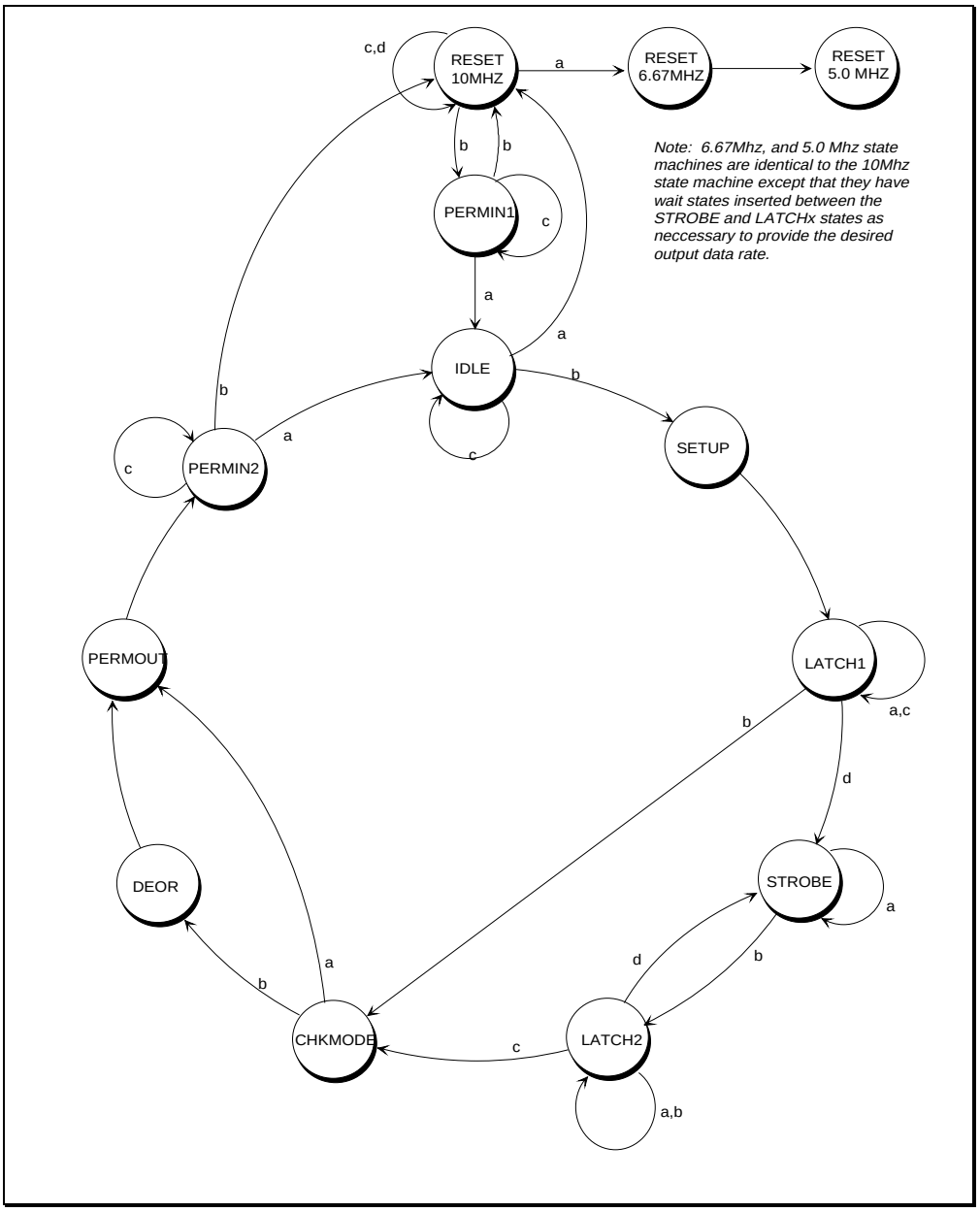


Figure 9 Output Port (OPORT) State Machine Diagram

3.3.2 PC4b OPORT State Machine Pseudo Listing

RESET:
a) IF mode=rate select THEN GOTO RESET 6.67
b) ELSEIF mode=Token_First
OR mode=Token_Middle and PERMIT_IN
OR mode=Token_Last and PERMIT_IN
OR mode=Token_Only
THEN status=1 GOTO PERMIN1
c) ELSEIF mode<>Disabled THEN status=1 GOTO RESET
d) ELSE GOTO RESET

PERMIN1:
a) IF /PERMIT_IN
OR mode=Token_First
OR mode=Token_Only
THEN status=2, Strobe driver enabled GOTO IDLE
b) ELSEIF mode=disabled THEN status=0 GOTO RESET
c) ELSE status=1 GOTO PERMIN1

IDLE:
a) IF mode=disabled THEN status=0 GOTO RESET
b) ELSEIF CFIFO not empty THEN status=3, data drivers enabled, strobe driver enabled, CFIFO read group select 1 GOTO SETUP
c) ELSE status=2, data drivers enabled, Strobe driver enabled, Clear Output Data Latch, GOTO IDLE

SETUP:
status=3, latch data, Data FIFO read, data drivers enabled, Strobe driver enabled, group select 1
CONTINUE

status=3, Data FIFO read, data drivers enabled, Strobe driver enabled, group select 1 GOTO LATCH1

LATCH1:
a) IF wait THEN status=4, Data FIFO read, data drivers enabled, Strobe driver enabled, group select 1 GOTO LATCH1
b) ELSEIF mode=Suppress Zero Event THEN status=3, data drivers enabled, Strobe driver enabled, CONTINUE
c) ELSEIF Data FIFO empty THEN status=5, Data FIFO read, data drivers enabled, Strobe driver enabled, group select 1 GOTO LATCH1
d) ELSE status=3, Data FIFO read, Data Strobe, data drivers enabled, Strobe driver enabled GOTO STROBE

status=3, data drivers enabled, Strobe driver enabled GOTO CHKMODE

STROBE:
a) IF mode=disabled THEN status=6, latch data, data drivers enabled, Strobe driver enabled GOTO STROBE
b) ELSE status=3, latch data, data drivers enabled, Strobe driver enabled GOTO LATCH2

LATCH2:
a) IF wait THEN status=4, data drivers enabled, Strobe driver enabled GOTO LATCH2
b) ELSEIF Data FIFO empty THEN status=5, data drivers enabled, Strobe driver enabled GOTO LATCH2
c) ELSEIF end of event flag THEN status=3, data latch clear, data drivers enabled, Strobe driver enabled GOTO CHKMODE
d) ELSE status=3, data strobe, Data FIFO read, data drivers enabled, Strobe driver enabled GOTO STROBE

CHKMODE:
a) IF mode=Token_First OR Token_Middle THEN status=3, Data FIFO read, Strobe driver enabled GOTO PERMOUT
b) ELSE status=3, Data FIFO read, Strobe driver enabled JUMP DEOR

DEOR: status=3 CONTINUE

status=3, end_of_record CONTINUE

status=3, end_of_record CONTINUE

status=3, end_of_record CONTINUE

status=3, end_of_record GOTO PERMOUT

PERMOUT: status=3 CONTINUE

status=3, PERMIT_OUT CONTINUE

status=3, PERMIT_OUT CONTINUE

status=3, PERMIT_OUT CONTINUE

status=3, PERMIT_OUT CONTINUE

status=3, PERMIT_OUT GOTO PERMIN2

PERMIN2:
a) IF PERMIT_IN
OR mode=Token_Only
THEN status=1 GOTO PIN2
b) ELSEIF mode=disabled THEN status=0 GOTO RESET
c) ELSE status=1 GOTO PERMIN2

PIN2:
a) IF /PERMIT_IN
OR mode=Token_Only
THEN status=1, Strobe driver enable GOTO IDLE
b) ELSEIF mode=disabled THEN status=0 GOTO RESET
c) ELSE status=1 GOTO PIN2

3.3.3 OPORT input/output signals

The processor can control and monitor the OPORT through the eight bit OPORT interface. The following signals are used to configure and monitor the OPORT.

RESET: This asynchronous line connects directly to the OPORT controller's hardware reset line. When held in reset, (low = reset) the OPORT controller disables its output drivers, ignores Permit tokens and mode selections.

MODE 2-0: OPORT Controller mode select lines.

The OPORT controller drives the following output lines to indicate status conditions to the processor:

MG_SEL: This line is used by the OPORT Controller's CPU interface to select which mode select group it should provide to the OPORT controller. By using the MG_SEL line in this way, the mode inputs to the OPORT controller are multiplexed to allow up to 16 modes to be defined instead of eight.

SM(0-3): Return the OPORT state machine status.

External interface:
PERMIT_OUT: Enable next device onto the token passing logical ring.
PERMIT_IN: Indicates the token has been received. This input is not latched. The OPORT controller must be initialized (set to an event mode) before it can recognize a Permit In.
STROBE: Signals active data on the pipeline latches, the rising edge is used to strobe that data into the personality card. Current personality cards invert the Strobe signal before using it to drive the RS-485 Strobe output.
WAIT: Pauses data transmission when true. WAIT is low true.

The OPORT is controlled from the CPU by programming the 8-bit control register in the OPORT interface. Refer to Output Port Controller interface for programming codes and status conditions.

3.3.4 OPORT Controller Interface

The output port controller interface (OPO_INTF) allows the CPU to control and monitor the output port microsequencer (OPORT). The CPU sets up the OPORT in one of the 8 defined modes and receives OPORT status information. The OPO_INTF can also drive an interrupt to the CPU system under some OPORT conditions.

Configuration:

The OPO_INTF is a byte wide port in the 68020 memory map. Its internal architecture consists of one control and two status registers. Commands are sent to the OPORT controller through the Control Register. The OPORT defined commands are: Set Mode, OPORT Reset, and Enable PERMIT_IN Interrupt. Register bit definitions are defined in Table 14. Refer to Figure 2 for the OPORTS base address.

Table 14 OPORT CPU Register Definitions

Address	Write Function	Read Function
OPORTS+0	None	Status register 1 **b0-b3**: OPORT Status **b4**: Control FIFO Status **b5**: Permit_In line flag **b6**: undefined **b7**: undefined
OPORTS+1	Control register **b0-b2:** OPORT Mode **b3**: OPORT hardware Reset (1=reset) **b4**: undefined **b5**: undefined **b6**: Permit_In_Mask **b7**: Suppress Zero Events (SZE)	Status register 2 **b0-b2**: OPORT Mode **B3:** OPORT hardware Reset **b4**: Permit_Out flag **b5** : undefined **b6**: Permit_In mask **b7**: Suppress Zero Events (SZE)

The software reset has the same effect over the OPORT as the hardware reset. The OPORT goes to the power on reset state, and the control FIFOS are cleared. The mode lines are cleared during a reset operation until a command word is written to the OPORT command register.

Note: When the Reset bit is set in the OPORT Control register, the OPORT Controller remains in Reset condition until the bit is cleared.

The interrupt condition is also reflected in the status registers, along with the OPORT state machine status. A latched version of the PERMIT_IN signal is available to the processor in OPORT Status Register 1. A read from this register clears the interrupt flags, and the PERMIN latch.

Table 15 OPORT Status Code Definitions

Status Code	Meaning
0	Hold/Idle 10 MHz (no mode set, OPORT is off line)
1	Waiting for Permit In
2	Waiting for End Of Event from FPORT
3	Transmitting Data
4	Waiting (OPORT Wait input is true)
5	Waiting FSCC Data FIFO is empty
6	Output paused by user
7 - 13	reserved
14	Hold/Idle 6.67 MHz (no mode set, OPORT is off line)
15	Hold/Idle 5.0 MHz (no mode set, OPORT is off line)

Control FIFO Status bit
This bit reads as a 0 when the Control FIFO is empty, and as a 1 when the Control FIFO is not empty. Reading this bit allows the 68020 to see if there are any complete events queued in the FSCC's data FIFO's. The Control FIFO Status bit will be a 1 after an End Of Event instruction has been executed, and before the OPORT starts outputting the event. This will be true if the PERMIT IN token has not been received, or the OPORT is busy outputting a previous event.

Permit In Flag
This bit reads as a 1 when the Permit_In Mask bit is set, and after a Permit In pulse is received. The Permit In Flag latch is cleared by reading OPORT Status Register 1, or resetting the OPORT controller.

OPORT Mode 0-2
These three bits select the OPORT Controller's operating mode.

OPORT Reset
This high true bit (1=reset) maps directly to the OPORT Controller's hardware reset pin. Setting this bit high places the OPORT controller into reset. The bit must be cleared to take the OPORT controller out of reset before a mode can be set.

Permit_In_Mask
This bit enables the Permit In flag in OPORT Status Register 1, and also the AUXREQ interrupt (GPIP2). If desired, receipt of a Permit In can generated a processor interrupt, if the Permit_In_Mask bit is set, and GPIP2 is enabled in the 68901 interrupt controller.

Suppress Zero Events (SZE Bit)
Setting this bit causes the OPORT to suppress output of zero word events. If the Data FIFO contains a Null event (no data) when an End_Of_Event instruction is executed, the OPORT controller will normally generate a Word Count/Header word (with the word count value of zero), and output this one word before passing the token. In some systems, it is desirable to reduce the amount of meaningless data collected, such as an event of word count zero. PC4b modules are equipped with a feature to allow suppression of the output of these zero word events. When the SZE bit is set to a one, and an End_Of_Event instruction is executed with an empty Data FIFO, the OPORT controller detects the zero word event, and passes the token without outputting the zero valued Word Count/Header word. The SZE feature has no other effect on operation of the OPORT controller, and it works in a similar manner regardless of output rate setting, or OPORT mode setting. After Reset, the SZE bit's default state is zero (SZE disabled).

3.3.5 OPORT Output Waveforms

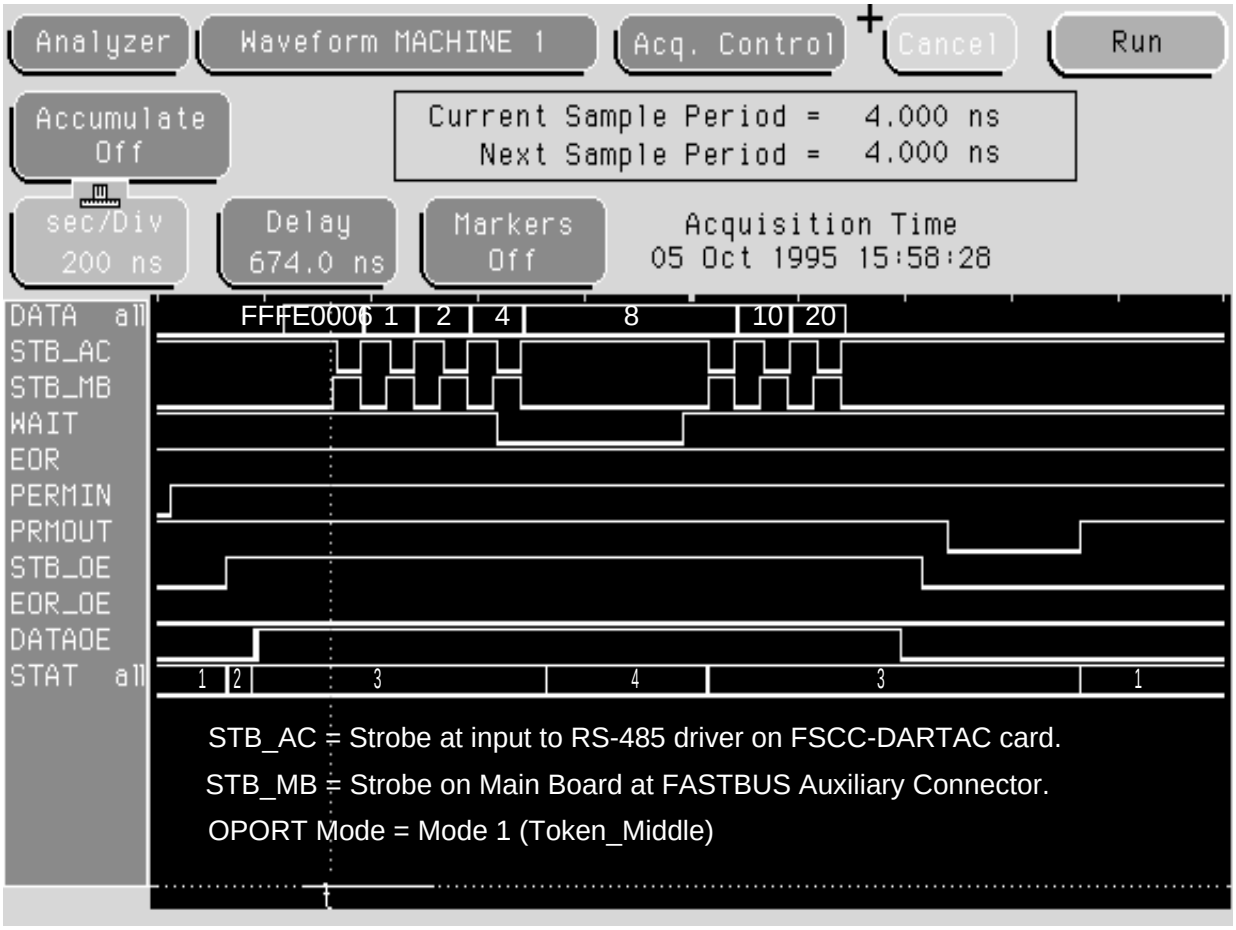


Figure 10 Output Port Token_Middle Analyzer Picture

Figure 10 is a Logic Analyzer snap-shot of a six word event as output through the Output Port (OPORT). The OPORT is configured in Token_Middle mode in this example. While waiting for the token (PERMIT_IN), the OPORT controller reports Status = 1 (Waiting for Permit). The Rising edge (the rising edge is the trailing edge, the falling edge is not shown) of PERMIT_IN causes the OPORT to enable the Strobe RS-485 driver (STB_OE goes true) and report Status = 2 (Waiting for End_Of_Event from FPORT). Since End_Of_Event has previously been executed by the FPORT controller, Status = 2 is only transitory as the OPORT controller begins outputting data. One cycle (50ns) after enabling the Strobe RS-485 driver, the data RS-485 drivers are enabled (DATAOE goes true), and the OPORT controller issues Status = 3 (Outputting Data). The first word out of the port is the header, containing the word count, error bit, and header bits. The header is zero, the low true Bad Event bit (bit 17) is high, and the undefined bits 18 through 31 are high in this example. The undefined bits in the Header/Word Count word are not guaranteed to be in a known state. Data is valid on the falling edge of Strobe. WAIT is driven true by the receiving module during the third data word. The output is paused during the fourth data word, and Status = 4 (Waiting due to WAIT true) is issued. Data Strobe goes true for word four after the OPORT controller sees WAIT go inactive. Status = 3 is again issued. After the last word in the event is output, the data RS-485 drivers are disabled, then the Strobe RS-485 driver is disabled. PERMIT_OUT is output to pass the token to the next FSCC (or other DART data source) in the token passing chain. Status = 1 is then issued. The EOR RS-485 driver is disabled in Token_Middle Mode.

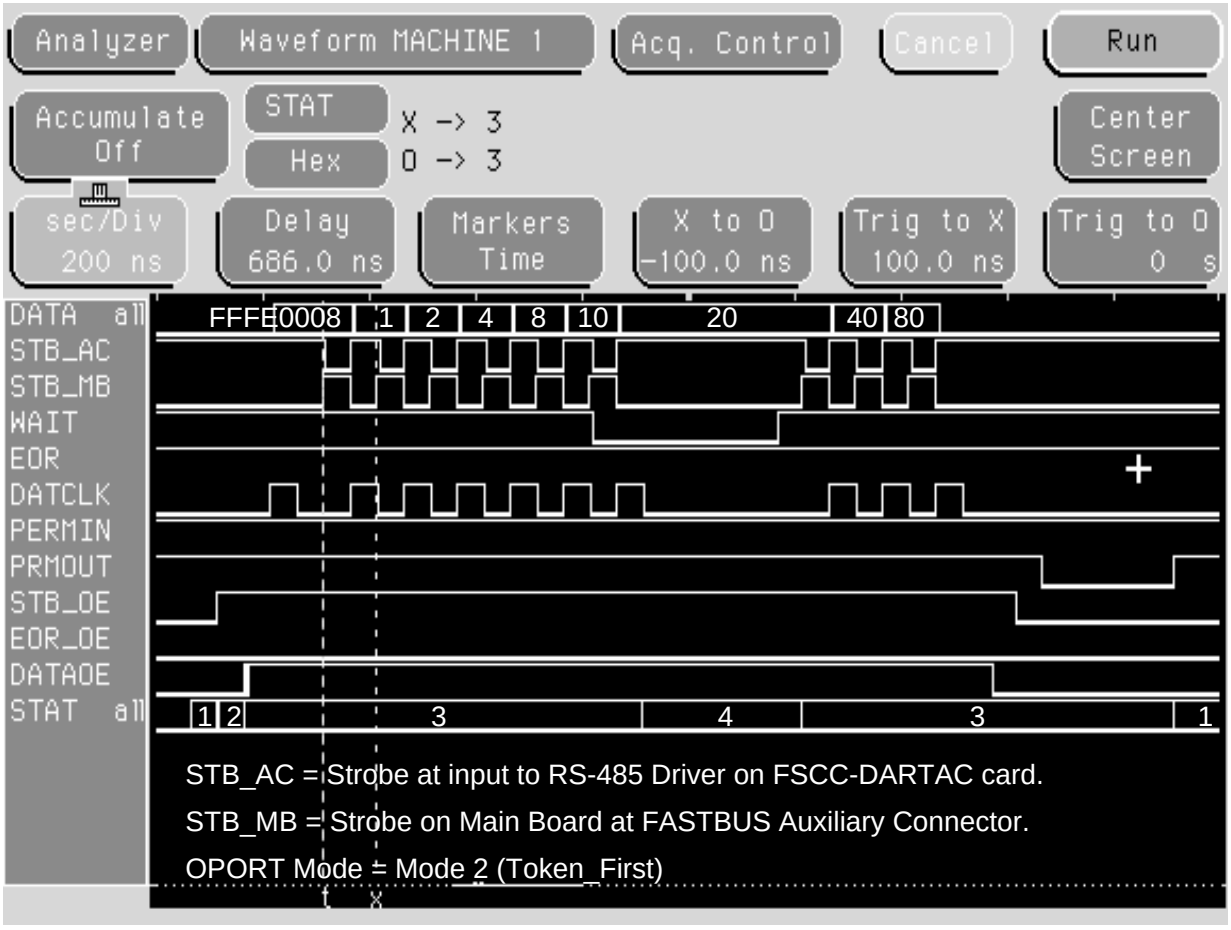


Figure 11 Output Port Token_First Analyzer Picture

Figure 11 shows a small event being output in Token_First mode. This mode is similar to Token_Middle mode above, with the exception being that the event is output without waiting for PERMIT_IN the first time after configuration.

The internal FSCC signal DATCLK is also shown in this figure. DATCLK is the clock signal to the output data latch. The output data latch is clocked by the rising edge of this signal. The data can be seen changing shortly after the rising edge of DATCLK.

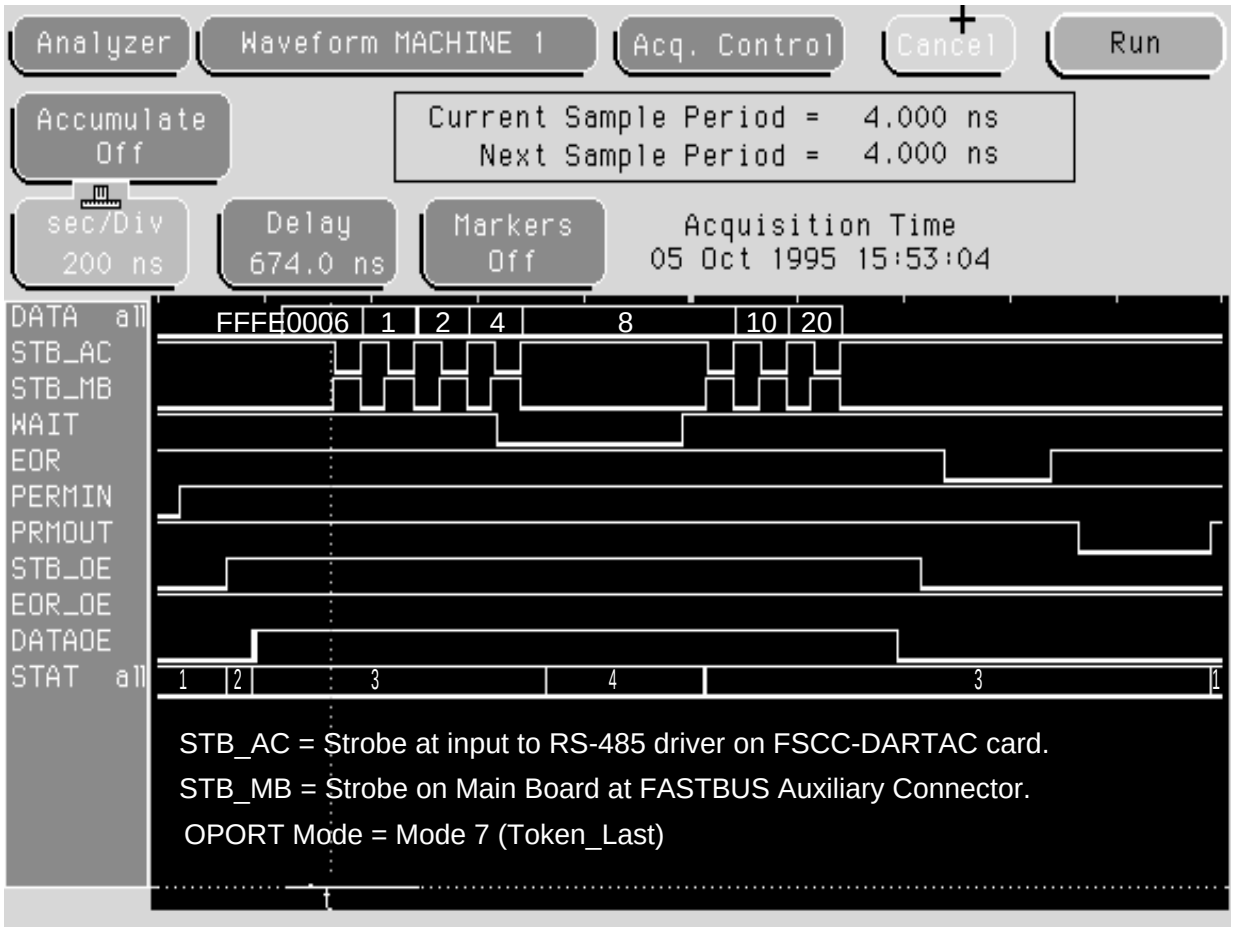


Figure 12 Output Port Token_Last Analyzer Picture

Figure 12 shows the Output Port operating in Token_Last mode. In Token_Last mode, the port functions similarly to Token_Middle mode described above, except that the End_Of_Record (EOR) signal is generated. The EOR RS-485 driver is also enabled (EOR_OE is true) whenever the OPORT is in Token_Last or Token_Only modes.

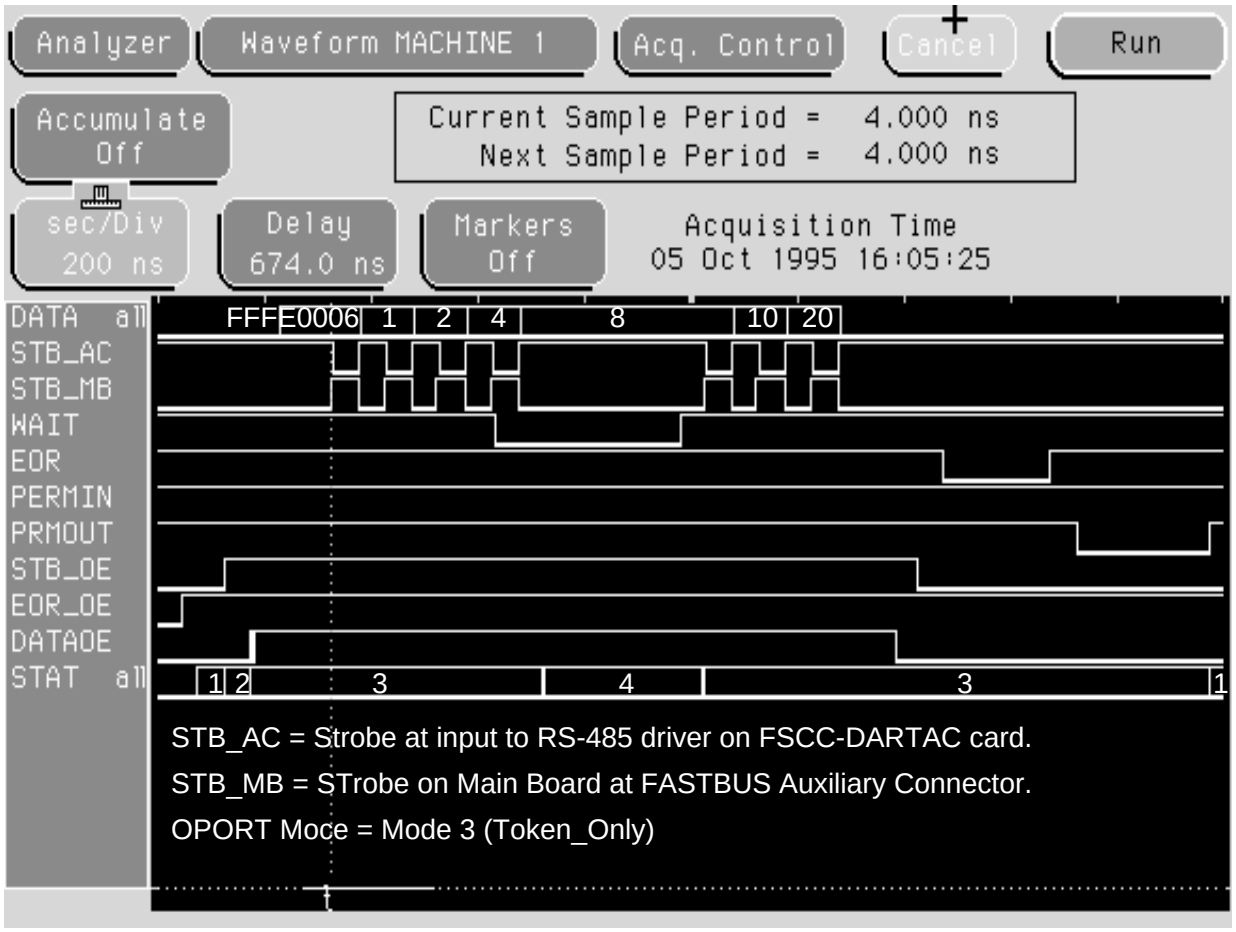


Figure 13 Output Port Token_Only Analyzer Picture

In Figure 13 Token_Only mode is shown. This mode is similar to Token_Last mode, except that the OPORT Controller does not wait for PERMIT_IN before beginning the data transfer. The EOR RS-485 driver is enabled (EOR_OE is true) and EOR is driven in Token_Only mode.

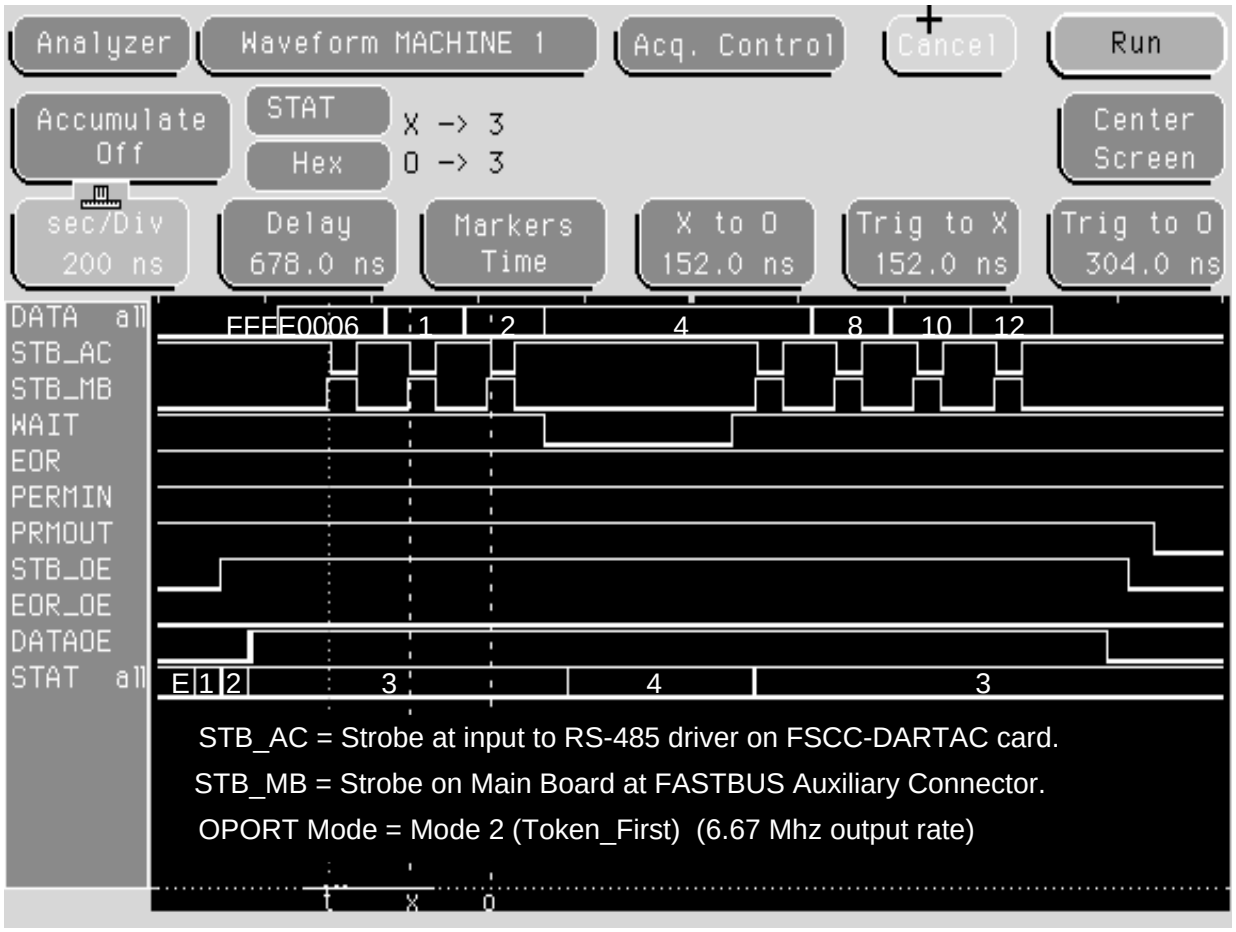


Figure 14 Output Port 6.67 MHz Analyzer Picture

Figure 14 shows the OPORT in Token_First mode after having previously been configured to output data at a 6.67 MHz rate (150ns per 32-bit word). Wherever the status reported to the processor would have been zero if configured to output at a 10MHz rate, it is hex E (decimal 14) in 6.67 MHz mode.

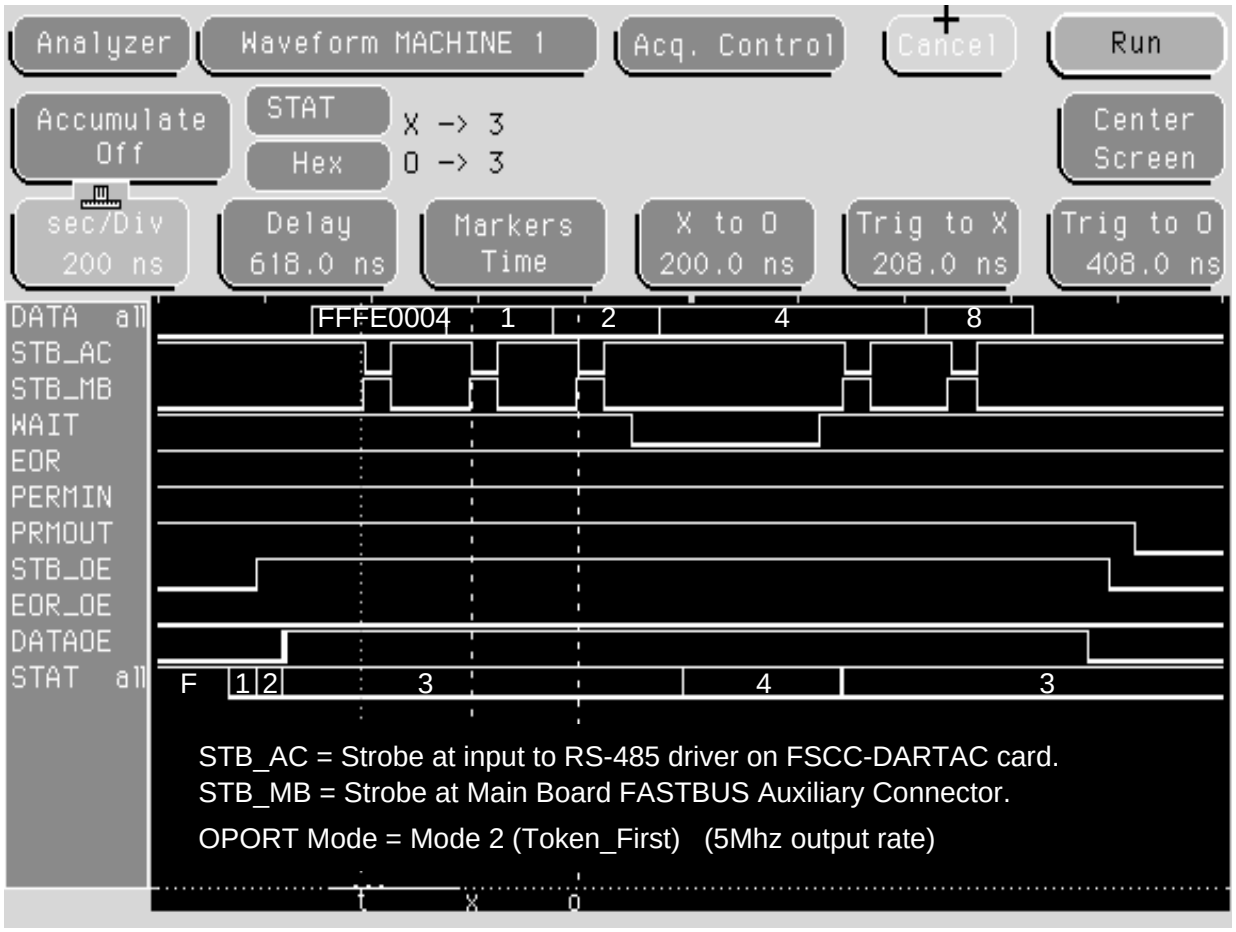


Figure 15 Output Port 5.0 MHz Analyzer Picture

Figure 15 shows the OPORT in Token_First mode after having previously been configured to output data at a 5.0 MHz rate (200ns per 32-bit word). Wherever the status reported to the processor would have been zero if configured to output at a 10 MHz rate, it is hex F (decimal 15) in 5.0 MHz mode.

3.3.6 OPORT Auxiliary Parallel Port

PC4b FSCC's have seven TTL lines connecting the personality card to the processor, in addition to the five control lines which connect directly to the OPORT Controller. The seven lines are connected to the 68020 via an EPLD. Currently this device is programmed to drive these seven lines as parallel outputs from an eight bit parallel port (see PARLLS in Figure 2). Note that currently available personality cards do not use the OPORT Aux. lines. Table 16 shows the bit definitions for the OPORT AUX. port. Pin assignments for the OPORT AUX. lines are shown in Table 4.

Table 16 OPORT Auxiliary Parallel Port Bit Definitions

Address	Write Function	Read Function
PARLLS+0	OPORT Aux. Output Latch **b0-b3**: OPORT Aux. bits 0 - 3 **b4:** reserved **b5-b7:** OPORT Aux. bits 5 - 7	OPORT Aux. Input Register **b0-b3**: OPORT Aux. bits 0 - 3 **b4**: reserved **b5-b7:** OPORT Aux. bits 5 - 7

3.3.7 Header and Event Counter Control System (H&C Controller)

3.3.7.1 Header and Counter (H&C)

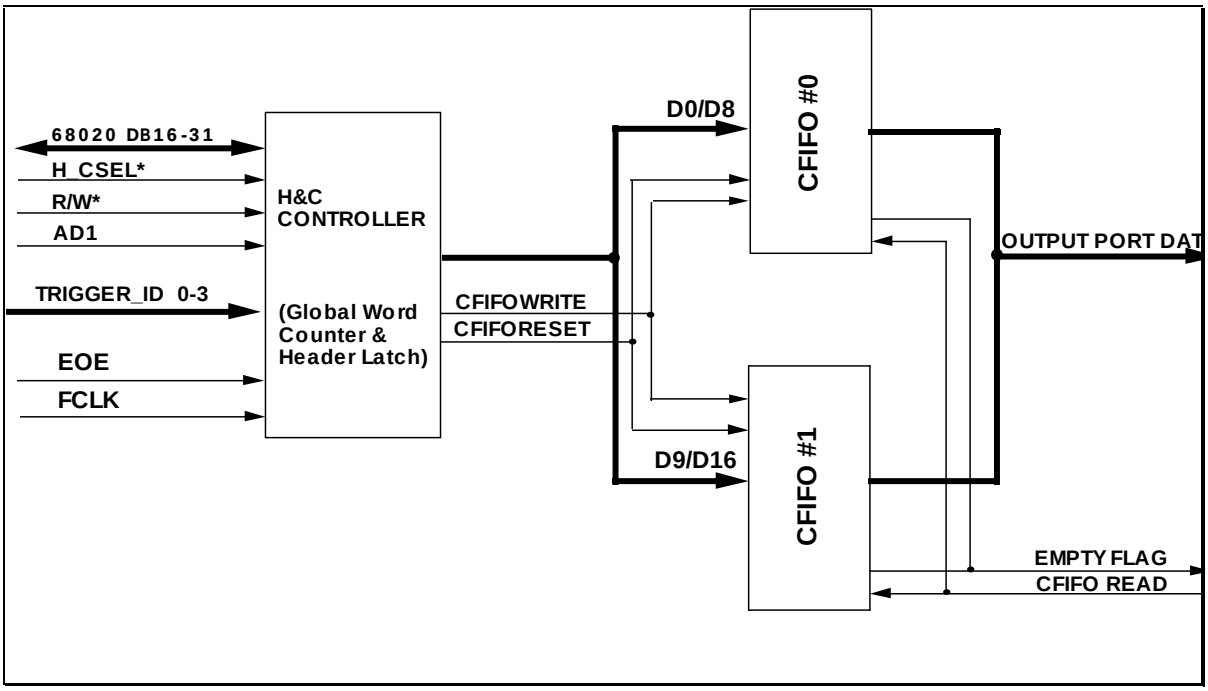
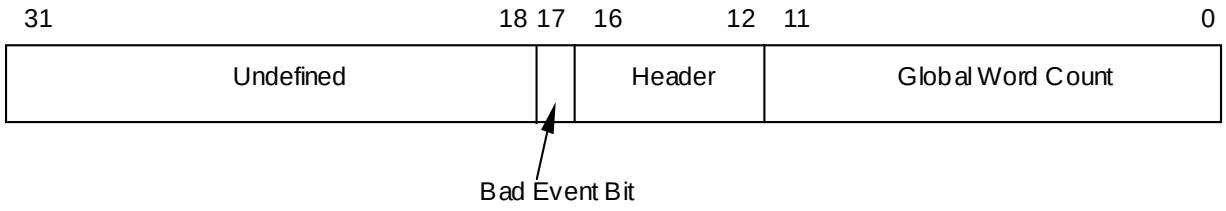


Figure 16 Header And Counter Block Diagram

The H&C system keeps a crate wide total count of 32-bit data words written into the Data FIFO. This "Global Word Counter" (GWC) includes data read from FASTBUS, and data written directly into the Data FIFO by the processor. The header is a 5-bit word which is either loaded by the processor, or loaded directly by the Trigger Input port into the Header Field. The 12 bit word counter is incremented by one, for each word transferred into the Data FIFO via a clock signal (FCLK) coming from the FASTBUS controller. When an End Of Event instruction is executed by the FPORT controller, the 5 header bits, the 12 counter bits, and the Bad Event bit (0=bad,1=OK), are packed into one 32-bit word and pushed into the Control FIFO. The Counter is then automatically reset to be ready for the next event. This H&C word then becomes the first word in the next data block to be transmitted out of the OPORT. The separate Control and Data FIFO's are necessary, since the FSCC does not know how many data words there are, until it is finished counting them as it reads them out of the front-end modules. Even though the H&C word is the last piece of data in a data block to be generated, it must go to the head of the line, to be output first. Therefore, the H&C word cannot go into the Data FIFO since it would then be after the data, and it would be output last. With the separate Control FIFO, the H&C word can be pulled out of the Control FIFO and output first, then the data can be clocked out of the Data FIFO to be output next.



Global Word Count and Header Word (H&C Word)

PC4b boards are equipped with a feature which allows the Trigger ID from the Front Panel Trigger Input port to be written automatically into the four least significant bits of the Header field. This feature eliminates the need for the 68020 to read the Trigger ID out of the Trigger port, and then write the value into the Header register. The default configuration for the Header, is that the 5 Header Register bits are written into header field of the header word upon execution of an End Of Event instruction. If the ETH (Enable Trigger in Header) bit is set in the H&C Control register, the Trigger ID bits clocked into the Front Panel Trigger port by the Trigger Strobe are written into low four bits of the header field.

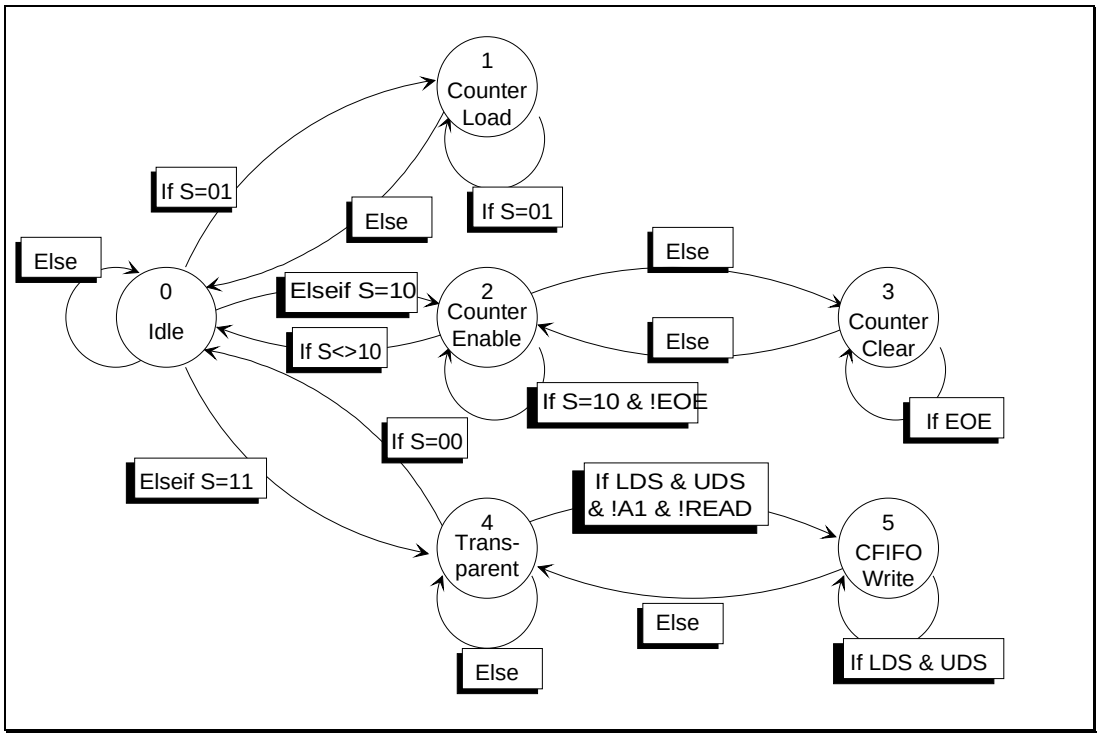


Figure 17 Header and Counter State Machine Diagram

3.3.7.2 System Interface

The entire Header, Global Word Counter and H&C control state machine is implemented in one EPM5128. The CPU sees the H&C system as a 16/8 bit peripheral. Four registers can be accessed from the processor bus. The counter and other H&C registers are accessible in bytes or words. The internal registers are selected by the 68020 through select and control lines: H&CSEL*, R/W*, UDS*, LDS* and A1. Table 17 gives a register select truth table for the H&C Controller.

Table 17 Header and Counter Control Signal Truth Table

A1	UDS	LDS	R/W	Reg. Selected
0	0	0	X	12 bit counter
0	0	1	X	Lower 8 bits of the Counter
0	1	0	X	Upper 4 bits of the counter
1	0	1	X	Header
1	1	0	0	Command
1	1	0	1	Status

The interface between the H&C controller and the FASTBUS (FPORT) sequencer is through FCLK (counter clock) and CEOE (Control End of Event) lines. Both are asynchronous to the 20 MHz clock of the state machine. FCLK increments H&C counter when it is in Increment mode. CEOE indicates that counting is over and the H&C value should be transferred to the CFIFO. The GWC is then reset to zero.

3.3.7.3 H&C Register Definitions

The H&CSEL base address is listed in Figure 2. Table 18 lists the register definitions for the H&C Controller. The GWC preload register is provided, but under normal use, preloading of the GWC is not necessary. The GWC is reset whenever and End_Of_Event instruction is executed.

Table 18 Header and Counter Register Map

Address	Write Function	Read Function
H&CSEL+0 (16-bit)	**GWC Preload register** b0-b11: Preload count b12-b15: Undefined	**GWC Register** b0-b11: Counter contents b12-b15: Undefined
H&CSEL+2 (8-bit)	**Header register** b0-b4: H0-4 Header preload b5-b7: Undefined	**Header register** b0-b4: H0-4 Header contents b5-b7: Undefined
H&CSEL+3 (8-bit)	**Control register** b0-b1: S0,S1 Command select field b2: H&C RESET b3-b6: Undefined b7: ETH control bit	**Status register** b0-b1: S0,S1 Command select field b2: Undefined b3-b6: Q0-Q3 H&C state status b7: ETH control bit

3.3.7.3.1 GWC Preload Register / GWC Register

A Write to this register loads the GWC Preload register. The lower 12 bits of the GWC Preload Register are transferred into the GWC when the Preload command is written into the H&C Control Register. Reading this register provides the current contents of the GWC.

3.3.7.3.2 Header Register

Bits 12-16 of the H&C Word contain the Event Header Field. These five bits are provided to allow identification tags to be attached to individual events. The Header Field (as well as the GWC field) of the H&C word is written into the CFIFO when an End_Of_Event instruction is executed by the FPORT. There are two modes of operation for the Header. In the default mode, the Header Field is filled by the value written into the Header Register by the CPU. The contents of this register may be changed at any time, but the user must be careful to know which event's ID tag is being attached to which event (the value in the Header Register when an End Of Event instruction is executed will be the tag attached to the event). In ETH (Enable Trigger ID in Header) mode, the contents of the lower four bits of the Header Field are filled by the value clocked into the Front Panel Trigger Input Port by the Trigger Strobe. These values are clocked into the Trigger Input FIFO. The FPORT clocks the Trigger values out of the Trigger FIFO with each End Of Event Instruction, so the Trigger Values automatically stay in sync with the event data in the Data FIFO. The Trigger Input value present when the Trigger Strobe is clocked is attached to the event read out for that Trigger. In this mode, bit five of the Header Field is filled by bit five of the Header Register.

3.3.7.3.3 Control Register

Bits b0 and b1 are the H&C Mode select bits S0 and S1. Mode definitions are described in Table 19.

Table 19 Header and Counter Mode Definitions

Control Mode	S1,S0	Function
HOLD:	00b	Global Word Counter does not increment.
INCRM:	10b	Global Word Counter increments one count for each data word written into the Data FIFO. Upon execution of an End Of Event instruction by the FPORT, the GWC (bits 0-11) is combined with the Header (bits 12-16) and the Bad Event Bit (bit 17), and pushed into the Control FIFO. The GWC is then cleared.
PRELOAD:	01b	When set to preload mode, data in the GWC Preload register is transferred to the Global Word Counter.
TRANSPARENT:	11b	Data written to the GWC Preload Register is written directly into the Control FIFO.

H&C RESET
This high true bit places the H&C into reset state when set. The Command bits are cleared (placed into Hold mode), the ETH bit is cleared, the GWC Preload Register is set to zero, and the GWC is set to zero. The H&C RESET bit must be cleared to take the H&C out of reset state.

ETH (Enable Trigger ID in Header) Control Bit
When set to a one, the Trigger ID value in the Trigger FIFO will be written directly into the low four bits of the Header field of the Header and Counter word when an End Of Event instruction is executed. This eliminates the need for the processor to read the Trigger ID port and then write this value into the Header register so that the Trigger ID may attached to the data block. The End Of Event instruction also clocks the Trigger FIFO so that the next trigger ID value will be ready when the next End Of Event instruction is executed. When the ETH bit is set to a zero, the 5-bit value in the Header Register (written by the processor) will be transferred into the Header and Counter word. Since the Trigger ID is 4 bits wide, and the Header field is 5 bits wide, only the four lower bits of the Header field are written by the Trigger ID. The fifth bit (Header bit 4) is always written by bit 4 of the Header Register. The default value of the ETH Control bit is zero.

3.3.7.3.4 Status Register

S1,S0
Command Select bits.

H&C state status
These bits reflect the current state number of the H&C Controller's state machine shown in Figure 17. They are provided mostly for diagnostic purposes.

ETH (Enable Trigger ID in Header) Control Bit
The state of the ETH control bit is reflected.

3.3.8 OPORT Auxiliary Connector Interface

All levels at the output port are single ended TTL. Level adapters to different protocols will be mounted on the FASTBUS auxiliary card. The data path is 32 bits wide. Two control lines are provided to regulate data flow:

WAIT pauses the OPORT
STROBE is a synchronous data strobe

The OPORT generates the pipeline latch clock, the output data strobe, and the output latch tri-state enable. Timing diagrams are shown in Figure 10 through Figure 15.

3.4 Communication Protocols

The following control line assignments apply to the RS-485 FSCC-DARTAC interface;

FBAUX pin B60	AC12	Strobe RS-485 driver Output Enable
FBAUX pin B59	AC11	32-bit Data driver Output Enable
FBAUX pin B58	AC10	WAIT* input
FBAUX pin B15	AC09	End Of Record
FBAUX pin B14	AC08	STROBE* output
FBAUX pin B4	AC00	not used (OPORT Aux. parallel port bit 0)
FBAUX pin B5	AC01	not used (OPORT Aux. parallel port bit 1)
FBAUX pin B6	AC02	not used (OPORT Aux. parallel port bit 2)
FBAUX pin B7	AC03	not used (OPORT Aux. parallel port bit 3)
FBAUX pin B8	AC04	End Of Record driver Output Enable
FBAUX pin B11	AC05	not used (OPORT Aux. parallel port bit 5)
FBAUX pin B12	AC06	not used (OPORT Aux. parallel port bit 6)
FBAUX pin B13	AC07	not used (OPORT Aux. parallel port bit 7)

4. Appendix A - FPORT Controller Instruction Set

4.1 FPORT Controller Normal Mode Instruction Set

4.1.1 BUS_ARBITRATE

BUS_ARBITRATE	SLOWBASE+$300

Description: Arbitrate for FASTBUS using the low byte of the data operand. Bits 0-5 supply the arbitration vector. Bit 7 enables assured access mode. Bit 6 (prioritized access mode) is ignored. Note that the data operand is a long word and is normally identical to the value of CSR 8.

Example Syntax: MOVE.L CSR_8, BUS_ARBITRATE

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: if IRQ(SEQINT) {goto INTS;}
elseif IGK(FRDY) return processor acknowledge A; FDSACK*⇓
else {request bus; FREQ⇑ goto C4;}
C5: return sequencer status;
return processor acknowledge B; FDSACK*⇓
C6: Delay Cycle; /*FPORT deselect*/
C7: Delay Cycle;
C8: Delay Cycle;

Note: In a multi-master system, the processor should examine the parallel port FSLV*, FRDY and FRAK inputs to confirm that the FSCC has either acquired the bus or been addressed as a slave while attempting to acquire the bus. 0

4.1.2 BUS_RELEASE

BUS_RELEASE	FASTBASE+$004

Description: Release FASTBUS arbitration lock.

Example Syntax: MOVE.L DUMMY, BUS_RELEASE

Operation:
C1: FPORT select;
return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: {DS=0; FCDS⇑
DK=0; FCDK⇑
AS=0; FCAS⇑
release bus;} FREL⇑
C5: return sequencer status;
C6: Delay Cycle; /*FPORT deselect*/
C7: Delay Cycle;
C8: Delay Cycle;

4.1.3 ADDRESS_DATA_GEOGRAPHICAL

ADDRESS_DATA_GEOGRAPHICAL	FASTBASE+$304

Description: Perform a FASTBUS geographical primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR, ADDRESS_DATA_GEOGRAPHICAL

Operation:

```
C1:   FPORT select;
      latch FB_ADDR; DCPBA⇑
      return processor acknowledge A; FDSACK*⇓
C2:   FPORT instruction fetch;
      return processor acknowledge B; FDSACK*⇓
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTF;
      elseif  WT(FRWT)    {reset short timer; TIMER⇓
                          goto C4;}
      elseif  AK(FRAK)    {reset short timer; TIMER⇓
                          MS=0;
                          return sequencer status;
                          exit;}
      else                {AS=1; FSAS⇑
                          RD=0;
                          MS=0;
                          EG=1; FDEG⇑
                          enable short timer;
                          goto C4;} TIMER⇑
```

4.1.4 ADDRESS_CSR_GEOGRAPHICAL

ADDRESS_CSR_GEOGRAPHICAL	FASTBASE+$308

Description: Perform a FASTBUS geographical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_CSR_GEOGRAPHICAL

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
        return processor acknowledge A; FDSACK*⇓
C2:     FPORT instruction fetch;
        return processor acknowledge B; FDSACK*⇓
C3:     instruction dispatch;
C4:     if      IRQ(SEQINT) goto INTF;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                            goto C4;}
        elseif  AK(FRAK)    {reset short timer; TIMER⇓
                            MS=0; FDMS0⇓
                            return sequencer status;
                            exit;}
        else                {AS=1; FSAS⇑
                            RD=0;
                            MS=1; FDMS0⇑
                            EG=1; FDEG⇑
                            enable short timer; TIMER⇑
                            goto C4;}
```

4.1.5 ADDRESS_DATA_LOGICAL

ADDRESS_DATA_LOGICAL	FASTBASE+$30C

Description: Perform a FASTBUS logical primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_DATA_LOGICAL

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
        return processor acknowledge A; FDSACK*⇓
C2:     FPORT instruction fetch;
        return processor acknowledge B; FDSACK*⇓
C3:     instruction dispatch;
C4:     if      IRQ(SEQINT) goto INTF;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                            goto C4;}
        elseif  AK(FRAK)    {reset short timer; TIMER⇓
                            MS=0;
                            return sequencer status;
                            exit;}
        else                {AS=1; FSAS⇑
                            RD=0;
                            enable short timer; TIMER⇑
                            goto C4;}
```

4.1.6 ADDRESS_CSR_LOGICAL

ADDRESS_CSR_LOGICAL	FASTBASE+$310

Description: Perform a FASTBUS logical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_CSR_LOGICAL

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
        return processor acknowledge A; FDSACK*⇓
C2:     FPORT instruction fetch;
        return processor acknowledge B; FDSACK*⇓
C3:     instruction dispatch;
C4:     if      IRQ(SEQINT) goto INTF;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                            goto C4;}
        elseif  AK(FRAK)    {reset short timer; TIMER⇓
                            MS=0; FDMS0⇓
                            return sequencer status;
                            exit;}
        else                {AS=1; FSAS⇑
                            RD=0;
                            MS=1; FDMS0⇑
                            enable short timer; TIMER⇑
                            goto C4;}
```

4.1.7 ADDRESS_DATA_BROADCAST

ADDRESS_DATA_BROADCAST	FASTBASE+$314

Description: Perform a FASTBUS broadcast primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_DATA_BROADCAST

Operation:

```
C1:   FPORT select;
      latch FB_ADDR; DCPBA⇑
      return processor acknowledge A; FDSACK*⇓
C2:   FPORT instruction fetch;
      return processor acknowledge B; FDSACK*⇓
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTF;
      elseif  WT(FRWT)    {reset short timer; TIMER⇓
                goto C4;}
      elseif  AK(FRAK)    {reset short timer; TIMER⇓
                MS=0; FDMS1⇓
                return sequencer status;
                exit;}
      else                {AS=1; FSAS⇑
                RD=0;
                MS=2; FDMS1⇑
                enable short timer;} TIMER⇑
```

4.1.8 ADDRESS_CSR_BROADCAST

ADDRESS_CSR_BROADCAST	FASTBASE+$318

Description: Perform a FASTBUS geographical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_CSR_BROADCAST

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
        return processor acknowledge A; FDSACK*⇓
C2:     FPORT instruction fetch;
        return processor acknowledge B; FDSACK*⇓
C3:     instruction dispatch;
C4:     if      IRQ(SEQINT) goto INTF;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                            goto C4;}
        elseif  AK(FRAK)    {reset short timer; TIMER⇓
                            MS=0; FDMS0⇓, FDMS1⇓
                            return sequencer status;
                            exit;}
        else                {AS=1; FSAS⇑
                            RD=0;
                            MS=3; FDMS0⇑, FDMS1⇑
                            enable short timer; TIMER⇑
                            goto C4;}
```

4.1.9 ADDRESS_RELEASE

ADDRESS_RELEASE	FASTBASE+$31C

Description: Release address lock.

Example Syntax: MOVE.L DUMMY,ADDRESS_RELEASE

Operation:

```
C1:     FPORT select;
        return processor acknowledge A; FDSACK*⇓
C2:     FPORT instruction fetch;
        return processor acknowledge B; FDSACK*⇓
C3:     instruction dispatch;
C4:     if      IRQ(SEQINT) goto INTF;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                            goto C4;}
        else                {DS=0; FCDS⇑
                            DK=0; FCDK⇑
                            MS=0; FDMS0⇓, FDMS1⇓,FDMS2⇓
                            AS=0; FSAS⇑
                            enable short timer;} TIMER⇑

C5:     if          IRQ(SEQINT) goto INTF;
        elseif  AK(FRAK)    goto C5;
        else                return sequencer status;
                            exit;
```

4.1.10 DATA_PROCESSOR_RANDOM_READ

DATA_PROCESSOR_RANDOM_READ	SLOWBASE+$320

Description: Perform a FASTBUS single word read data cycle.

Example Syntax: MOVE.L DATA_PROCESSOR_RANDOM_READ,DATA

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTS;
      elseif WT(FRWT) {reset short timer; TIMER⇓
                  goto C4;}
      elseif  DK(FRDK)    return processor acknowledge A; FDSACK*⇓
      else          {enable short timer; TIMER⇑
                  RD=1; FDRD⇑
                  MS=0;
                  DS=1; FSDS⇑
                  goto C4;}
C5:   reset short timer; TIMER⇓
      return processor acknowledge B; FDSACK*⇓
C6:   if      IRQ(SEQINT) INTF;
      elseif  WT(FRWT)    {reset short timer; TIMER⇓
                  goto C6;}
      elseif  !DK(FRDK*)  {return sequencer status;
                  reset short timer; TIMER⇓}
      else          {enable short timer; TIMER⇑
                  RD=0; FDRD⇓
                  MS=0;
                  DS=0; FCDS⇑
                  goto C6;}
C7:   delay cycle;  /* processor deselect */
C8:   delay cycle;
C9:   delay cycle;
```

4.1.11 DATA_PROCESSOR_RANDOM_WRITE

DATA_PROCESSOR_RANDOM_WRITE	FASTBASE+$324

Description: Perform a FASTBUS single word write data cycle.

Example Syntax: MOVE.L DATA,DATA_PROCESSOR_RANDOM_WRITE

Operation:

```
C1:   FPORT select;
      latch DATA; DCPBA⇑
      return processor acknowledge A; FDSACK*⇓
C2:   FPORT instruction fetch;
      return processor acknowledge B; FDSACK*⇓
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTF;
      elseif WT(FRWT) {reset short timer; TIMER⇓
                       goto C4;}
      elseif  DK(FRDK)    reset short timer; TIMER⇓
      else                {enable short timer; TIMER⇑
                           RD=0;
                           MS=0;
                           DS=1; FSDS⇑
                           goto C4;}
C5:   if      IRQ(SEQINT) goto INTF;
      elseif  WT(FRWT)    {reset short timer; TIMER⇓
                           goto C5;}
      elseif  !DK(FRDK*)  {return sequencer status;
                           exit;}
      else                {enable short timer; TIMER⇑
                           RD=0;
                           MS=0;
                           DS=0; FCDS⇑
                           goto C5;}
```

4.1.12 DATA_PROCESSOR_SEC_ADDRESS_READ

DATA_PROCESSOR_SEC_ADDRESS_READ	SLOWBASE+$328

Description: Perform a FASTBUS secondary address read cycle.

Example Syntax: MOVE.L DATA_PROCESSOR_SEC_ADDRESS_READ,SADDR

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTS;
      elseif WT(FRWT) {reset short timer; TIMER⇓
                  goto C4;}
      elseif  DK(FRDK)    return processor acknowledge A; FDSACK*⇓
      else              {enable short timer; TIMER⇑
                         RD=1; FDRD⇑
                         MS=2; FDMS1⇑
                         DS=1; FSDS⇑
                         goto C4;}
C5:   reset short timer; TIMER⇓
      return processor acknowledge B; FDSACK*⇓
C6:   if      IRQ(SEQINT) goto INTF;
      elseif  WT(FRWT)    {reset short timer; TIMER⇓
                  goto C6;}
      elseif  !DK(FRDK*)  {reset short timer; TIMER⇓
                  return sequencer status;}
      else              {enable short timer; TIMER⇑
                         RD=0; FDRD⇓
                         MS=2; FDMS1⇑
                         DS=0; FCDS⇑
                         goto C6;}
C7:   delay cycle;      /* processor deselect */
C8:   delay cycle;
C9:   delay cycle;
```

4.1.13 DATA_PROCESSOR_SEC_ADDRESS_WRITE

DATA_PROCESSOR_SEC_ADDRESS_WRITE	FASTBASE+$32C

Description: Perform a FASTBUS secondary address write cycle.

Example Syntax: MOVE.L SADDR,DATA_PROCESSOR_SEC_ADDRESS_WRITE

Operation:

```
C1:     FPORT select;
        latch SADDR; DCPBA⇑
        return processor acknowledge A; FDSACK*⇓
C2:     FPORT instruction fetch;
        return processor acknowledge B; FDSACK*⇓
C3:     instruction dispatch;
C4:     if      IRQ(SEQINT) goto INTF;
        elseif WT(FRWT) {reset short timer; TIMER⇓
                         goto C4;}
        elseif  DK(FRDK)    reset short timer; TIMER⇓
        else                {enable short timer; TIMER⇑
                             RD=0;
                             MS=2; FDMS1⇑
                             DS=1; FSDS⇑
                             goto C4;}
C5:     if      IRQ(SEQINT) goto INTF;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                             goto C5;}
        elseif  !DK(FRDK*)  {return sequencer status;
                             exit;}
        else                {enable short timer; TIMER⇑
                             RD=0;
                             MS=0; FDMS1⇓
                             DS=0; FCDS⇑
                             goto C5;}
```

4.1.14 DATA_PROCESSOR_BLOCK_TRANSFER_READ

DATA_PROCESSOR_BLOCK_TRANSFER_READ

SLOWBASE+$008

Description: Input one word of a FASTBUS block transfer read cycle. Processor Block transfer reads occur by executing one DATA_PROCESSOR_BLOCK_TRANSFER_READ instruction for each data word to be input. The FASTBUS sequencer maintains the proper state of DS after completion of the instruction. The DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE instruction must then be used to "clean-up" after a string of DATA_PROCESSOR_BLOCK_TRANSFER_READ instructions have been executed. Note that the state of the MS and RD lines are set to zero after each DATA_PROCESSOR_BLOCK_TRANSFER_READ instruction is executed. This is allowed by the FASTBUS specification, however, some slaves are confused by this behavior. Since the DATA_PROCESSOR_BLOCK_TRANSFER_READ instruction effectively leaves the state of the Sequencer, and of the attatched slave in the middle of a block transfer operation, care should be taken to ensure that a DATA_PROCESSOR_BLOCK_TRANSFER_READ instruction always follows any group of DATA_PROCESSOR_BLOCK_TRANSFER_READ instructions.

Example Syntax: MOVE.L DATA_PROCESSOR_BLOCK_TRANSFER_READ,DATA

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTS;
      elseif WT(FRWT) {reset short timer; TIMER⇓
                  goto C4;}
      elseif  ! DS(FRDS*)  {enable short timer; TIMER⇑
                  RD=1; FDRD⇑
                  MS=1; FDMS1⇑
                  DS=1; FSDS⇑
                  goto C6;}
      else              {enable short timer; TIMER⇑
                  RD=1; FDRD⇑
                  MS=1; FDMS0⇑
                  DS=0;} FCDS⇑
C5:   if      IRQ(SEQINT) goto INTS;
      elseif ! DK(FRDK*)    {return processor acknowledge A; FDSACK*⇓
                  reset short timer; TIMER⇓
                  goto C7;}
      else              goto C5;
C6:   if      IRQ(SEQINT) goto INTS;
      elseif  DK(FRDK)    {return processor acknowledge A; FDSACK*⇓
                  reset short timer;} TIMER⇓
```

```
        else            goto C6;
C7:     {return processor acknowledge B; FDSACK*⇓
                        return sequencer status;}
C8:     delay cycle;    /* processor deselect */
C9:     delay cycle;
C10: delay cycle;
C11: delay cycle;
```

4.1.15 DATA_PROCESSOR_BLOCK_TRANSFER_WRITE

DATA_PROCESSOR_BLOCK_TRANSFER_WRITE

FASTBASE+$00C

Description: Output one word of a FASTBUS block transfer write cycle. Since the FSCC has been optimized as a read-out controller, there is no FIFO to queue data to be output through the FASTBUS port. Block transfer writes occur by executing one DATA_PROCESSOR_BLOCK_TRANSFER_WRITE instruction for each data word to be output. The FASTBUS sequencer maintains the proper state of DS after completion of the instruction. The DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE instruction must then be used to "clean-up" after a string of DATA_PROCESSOR_BLOCK_TRANSFER_WRITE instructions have been executed. Note that the state of the MS lines are set to zero after each DATA_PROCESSOR_BLOCK_TRANSFER_WRITE instruction is executed. This is allowed by the FASTBUS specification, however, some slaves are confused by this behavior. Since the DATA_PROCESSOR_BLOCK_TRANSFER_WRITE instruction effectively leaves the state of the Sequencer, and of the attatched slave in the middle of a block transfer operation, care should be taken to ensure that a DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE instruction always follows any group of DATA_PROCESSOR_BLOCK_TRANSFER_WRITE instructions.

Example Syntax: MOVE.L DATA,DATA_PROCESSOR_BLOCK_TRANSFER_WRITE

Operation:

```
C1:   FPORT select;
      latch DATA; DCPBA⇑
      return processor acknowledge A; FDSACK*⇓
C2:   FPORT instruction fetch;
      return processor acknowledge B; FDSACK*⇓
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTF;
      elseif WT(FRWT) {reset short timer; TIMER⇓
                       RD=0;
                       MS=1; FDMS0⇑
                       goto C4;}
      elseif ! DS(FRDS*)  {enable short timer; TIMER⇑
                       RD=0;
                       MS=1; FDMS0⇑
                       DS=1; FSDS⇑
                       goto C6;}
      else             {enable short timer; TIMER⇑
                       RD=0;
                       MS=1; FDMS0⇑
                       DS=0;} FCDS⇑
C5:   if      IRQ(SEQINT) goto INTF;
      elseif ! DK(FRDK*)  {reset short timer; TIMER⇓
                       RD=0;
```

```
                                MS=0; FDMS0⇓
                                return sequencer status;
                                exit;}
        else                    goto C5;
C6:     if      (IRQ)           goto INTF;
        elseif  (DK)            {reset short timer; TIMER⇓
                                RD=0;
                                MS=0; FDMS0⇓
                                exit;}
        else                    goto C6;
```

4.1.16 DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE

DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE
FASTBASE+$330

Description: Perform termination step of a FASTBUS block transfer read or write operation. This instruction allows the state of DS to be set low after executing one or more DATA_PROCESSOR_BLOCK_TRANSFER_WRITE or DATA_PROCESSOR_BLOCK_TRANSFER_READ instructions. It should always be executed after a group of either of these two instructions have been executed.

Example Syntax: MOVE.L DUMMY,DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE

Operation:

```
C1:   FPORT select;
      return processor acknowledge A; FDSACK*⇓
C2:   FPORT instruction fetch;
      return processor acknowledge B; FDSACK*⇓
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTF;
      elseif WT(FRWT) goto C4;
      elseif  !FRDK  {return sequencer status; goto C5;}
      else            {MS=0; FDMS0⇓, FDMS1⇓, FDMS2⇓
                       RD=0; FDRD⇓
                       DS=0;} FCDS⇑
C5:   delay cycle;
C6:   delay cycle;
C7:   delay cycle;
```

4.1.17 DATA_FIFO_BLOCK_TRANSFER_READ

DATA_FIFO_BLOCK_TRANSFER_READ	FASTBASE+$334

Description: Perform a FASTBUS block transfer read to the Data FIFO. The block transfer is terminated upon receipt of SS=2 from the slave, or if the number of words specified in the Local Counter have been read (if enabled).

Example Syntax: MOVE.L DUMMY,DATA_FIFO_BLOCK_TRANSFER_READ

Operation:

```
C1:     FPORT select;
        return processor acknowledge A; FDSACK*⇓
C2:     FPORT instruction fetch;
        return processor acknowledge B; FDSACK*⇓
C3:     instruction dispatch;
C4:     if      IRQ(SEQINT) goto INTF;
        elseif WT(FRWT) {reset short timer; TIMER⇓
                         enable FIFO; DFIFOEN⇑,PFIFOEN⇑
                         RD=1; FDRD⇑
                         MS=1; FDMS0⇑
                         goto C4;}
        else            {enable FIFO; DFIFOEN⇑,PFIFOEN⇑
                         RD=1; FDRD⇑
                         MS=1; FDMS0⇑
                         DS=1; FSDS⇑
                         local counter mode = decrement; LC0⇑, LC1⇑}
C5:     {local counter mode = hold; LC0⇓, LC1⇓
        enable short timer;} TIMER⇑
C6:     continue;               /* delay cycle */
C7:     if      ((FRDK*FRSS1)#SEQINT #FEOB)  goto C11;
        elseif WT(FRWT) {reset short timer; TIMER⇓
                         goto C7;}
        elseif  DK(FRDK)    {DS=0; FCDS⇑
                         local counter mode = decrement; LC0⇑, LC1⇑
                         reset short timer; TIMER⇓
                         clock global word counter; FCLK⇑}
        else            {enable short timer; TIMER⇑
                         goto C7;}
C8:     {local counter mode = hold; LC0⇓, LC1⇓    /* delay cycle */
                         enable short timer; TIMER⇑}
C9: delay cycle;
C10: if ((!FRDK*FRSS1)#SEQINT#FEOB)      goto C11;
        elseif  WT(FRWT)    {reset short timer;
                         goto C10;}
        elseif  !DK(!FRDK)  {DS=1; FSDS⇑
                         local counter mode=decrement; LC0⇑, LC1⇑
                         reset short timer; TIMER⇓
                         clock global word counter; FCLK⇑
```

```
                              goto C5;}
          else                goto C10;
C11: if   SS2(FRSS1) clock global word counter; FCLK⇑
/* block transfer termination */
C12: if   IRQ(SEQINT)    goto INTF;
          elseifDS(FRDS) continue;
          else           goto C22;
/* termination routine for odd word count transfer */
/* check that DK is high */
/* set DS low and wait for DK low */
/* data written to FIFO on DK down is dummy word with EOE flag*/
C13:if    IRQ(SEQINT)    goto INTF;
          elseif    DK(FRDK)     {reset short timer; TIMER⇓
                              MS=0;} FDMS0⇓
          else              {enable short timer; TIMER⇑
                              goto C13;}
C14:continue;                     /* delay cycle */
C15:continue;                     /* delay cycle */
C16:continue;                     /* delay cycle */
C17:if    IRQ(SEQINT)    goto INTF;
          elseif    WT(FRWT)     {reset short timer; TIMER⇓
                              goto C17;}
          else              {enable short timer; TIMER⇑
                              DS=0;} FCDS⇑
C18:if    IRQ(SEQINT)    goto INTF;
          elseif    ! DK(! FRDK) {RD=0; FDRD⇓
                              reset short timer; TIMER⇓
                              return sequencer status}
          else              {enable short timer; TIMER⇑
                              goto C18;}
C19:continue;                     /* delay cycle */
C20:continue;                     /* delay cycle */
C21:exit;
/* termination routine for even word count transfer */
C22:if    IRQ(SEQINT)    goto INTF;    /* check that DK is low */
          elseif    ! DK(! FRDK) {RD=0; FDRD⇓
                              reset short timer; TIMER⇓
                              return sequencer status}
          else              {enable short timer; TIMER⇑
                              goto C22;}
C23:continue;                     /* delay cycle */
C24:continue;                     /* delay cycle */
C25:continue;                     /* delay cycle */
C25:exit;
```

4.1.18 DATA_FIFO_PIPELINED_READ_100

DATA_FIFO_PIPELINED_READ_100	FASTBASE+$338

Description: Perform a FASTBUS pipelined read to the Data FIFO at 100 nsec/word. This instruction functions similarly to the DATA_PROCESSOR_BLOCK_TRANSFER_READ instruction except that it follows the FASTBUS specification for Pipelined operations. Note that the FSCC does not implement a DK counter as called for in the FASTBUS specification.

Example Syntax: MOVE.L DUMMY,DATA_FIFO_PIPELINED_READ_100

Operation:

```
C1:   FPORT select;
      return processor acknowledge A; FDSACK*⇓
C2:   FPORT instruction fetch;
      return processor acknowledge B; FDSACK*⇓
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTF;
      elseif WT(FRWT) goto C4;
      elseif  EOB(FEOB)   goto PIPELINE_TERMINATE;
      else          {enable FIFO; PFFEN⇑, DFFEN⇑
                     RD=1; FDRD⇑
                     MS=3; FDMS0⇑,FDMS1⇑
                     DS=1; FSDS⇑
                     local counter mode = decrement; LC0⇑,LC1⇑
                     clock global word counter;} FCLK⇑
C5:   local counter mode = hold; LC0⇓,LC1⇓
C6:   if      IRQ(SEQINT) goto INTF;
      elseif WT(FRWT) goto C6;
      elseif  EOB(FEOB)   goto PIPELINE_TERMINATE;
      else          {DS=0; FCDS⇑
                     local counter mode = decrement; LC0⇑,LC1⇑
                     clock global word counter;} FCLK⇑
C7:   {local counter mode = hold; LC0⇓,LC1⇓
      goto C4;}
```

4.1.19 DATA_FIFO_PIPELINED_READ_200

DATA_FIFO_PIPELINED_READ_200	FASTBASE+$33C

Description: Perform a FASTBUS pipelined read to the Data FIFO at 200 nsec/word. This instruction functions similarly to the DATA_PROCESSOR_BLOCK_TRANSFER_READ instruction except that it follows the FASTBUS specification for Pipelined operations. Note that the FSCC does not implement a DK counter as called for in the FASTBUS specification.

Example Syntax: MOVE.L DUMMY,DATA_FIFO_PIPELINED_READ_200

Operation:

```
C1:   FPORT select;
      return processor acknowledge A; FDSACK*⇓
C2:   FPORT instruction fetch;
      return processor acknowledge B; FDSACK*⇓
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT)  goto INTF;
      elseif WT(FRWT) goto C4;
      elseif  EOB(FEOB)    goto PIPELINE_TERMINATE;
      else            {enable FIFO; PFFEN⇑, DFFEN⇑
                       RD=1; FDRD⇑
                       MS=3; FDMS0⇑,FDMS1⇑
                       DS=1; FSDS⇑
                       local counter mode = decrement; LC0⇑,LC1⇑
                       clock global word counter;} FCLK⇑
C5:   local counter mode = hold; LC0⇓,LC1⇓
C6:   continue;              /* delay cycle */
C7:   continue;              /* delay cycle */
C8:   if      IRQ(SEQINT)  goto INTF;
      elseif WT(FRWT) goto PIPELINE_TERMINATE;
      elseif  EOB(FEOB)    goto C12;
      else            {DS=0; FCDS⇑
                       local counter mode = decrement; LC0⇑,LC1⇑
                       clock global word counter;} FCLK⇑
C9:   local counter mode = hold; LC0⇓,LC1⇓
C10:continue;              /* delay cycle */
C11:goto C4;               /* delay cycle */
```

4.1.20 DATA_FIFO_PIPELINED_READ_400

DATA_FIFO_PIPELINED_READ_400	FASTBASE+$340

Description: Perform a FASTBUS pipelined read to the Data FIFO at 400 nsec/word. This instruction functions similarly to the DATA_PROCESSOR_BLOCK_TRANSFER_READ instruction except that it follows the FASTBUS specification for Pipelined operations. Note that the FSCC does not implement a DK counter as called for in the FASTBUS specification.

Example Syntax: MOVE.L DUMMY,DATA_FIFO_PIPELINED_READ_400

Operation:

```
C1:    FPORT select;
       return processor acknowledge A; FDSACK*⇓
C2:    FPORT instruction fetch;
       return processor acknowledge B; FDSACK*⇓
C3:    instruction dispatch;
C4:    if      IRQ(SEQINT)  goto INTF;
       elseif WT(FRWT) goto C4;
       elseif  EOB(FEOB)    goto PIPELINE_TERMINATE;
       else            {enable FIFO; PFFEN⇑, DFFEN⇑
                        RD=1; FDRD⇑
                        MS=3; FDMS0⇑,FDMS1⇑
                        DS=1; FSDS⇑
                        local counter mode = decrement; LC0⇑,LC1⇑
                        clock global word counter;} FCLK⇑
C5:    local counter mode = hold; LC0⇓,LC1⇓
C6:    continue;               /* delay cycle */
C7:    continue;               /* delay cycle */
C8:    continue;               /* delay cycle */
C9:    continue;               /* delay cycle */
C10:continue;                  /* delay cycle */
C11:continue;                  /* delay cycle */
C12:if  IRQ(SEQINT)    goto INTF;
       elseif WT(FRWT) goto C12;
       elseif  EOB(FEOB)    goto PIPELINE_TERMINATE;
       else            {DS=0; FCDS⇑
                        local counter mode = decrement; LC0⇑,LC1⇑
                        clock global word counter;} FCLK⇑
C13:local counter mode = hold; LC0⇓,LC1⇓
C14:continue;                  /* delay cycle */
C15:continue;                  /* delay cycle */
C16:continue;                  /* delay cycle */
C17:continue;                  /* delay cycle */
C18:continue;                  /* delay cycle */
C19:goto C4;                   /* delay cycle */
```

4.1.21 SEQUENCER_NULL

SEQUENCER_NULL	FASTBASE+$020

Description: Access the FPORT Controller without performing any operation. Confirms that the FPORT Controller is active. If the FPORT Controller is stalled, SEQUENCER_NULL will cause a Processor BUS ERROR interrupt.

Example Syntax: MOVE.L DUMMY,SEQUENCER_NULL

Operation:
C1: FPORT select;
return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: {return sequencer status;
exit;}

Note: If FASTBUS WT is asserted, the FPORT Controller will wait indefinitely for a slave response. A WT time-out will only be generated (after the long time-out period) if the processor attempts another FASTBUS instruction while the first operation is pending. In cases where the processor does not access the FPORT Controller for extended periods of time (e.g., a standalone microcode readout loop) an occasional SEQUENCER)NULL instruction will detect a FASTBUS lockup condition. SEQUENCER_NULL will also guarantee that all pending FASTBUS operations have been completed by clearing the instruction pipeline.

4.1.22 LOCAL_COUNTER_LOAD

LOCAL_COUNTER_LOAD	SLOWBASE+$010

Description: Initialize Local Word Counter for block transfer.

Example Syntax: MOVE.L COUNT,LOCAL_COUNTER_LOAD

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {return processor acknowledge A; FDSACK*⇓
local counter mode = load;} LC1⇑
C5: {return processor acknowledge B; FDSACK*⇓
local counter mode = hold;} LC1⇓
C6: return sequencer status; /* processor deselect */
C7: delay cycle;
C8: delay cycle

Note: The counter is loaded from the low order 12 bits of COUNT.

4.1.23 LOCAL_COUNTER_READ

LOCAL_COUNTER_READ	SLOWBASE+$014

Description: Read current value of Local Word Counter.

Example Syntax: MOVE.L LOCAL_COUNTER_READ,COUNT

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {return processor acknowledge A; FDSACK*⇓
local counter mode = read;} SLCOE⇑
C5: return processor acknowledge B; FDSACK*⇓
C6: return sequencer status; /* processor deselect */
C7: delay cycle;
C8: delay cycle;

Note: The counter is returned in the low order 12 bits of COUNT.

4.1.24 FIFO_WRITE

FIFO_WRITE	SLOWBASE+$018

Description: Write a single word from the processor to the Data FIFO.

Example Syntax: MOVE.L DATA,FIFO_WRITE

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {return processor acknowledge A; FDSACK*⇓
clock FIFO input;} SDW⇑
C5: {return processor acknowledge B; FDSACK*⇓
clock global word counter;} FCLK⇑
C6: return sequencer status; /* processor deselect */
C7: delay cycle;
C8: delay cycle;

4.1.25 END_OF_EVENT

END_OF_EVENT	FASTBASE+$024

Description: Write dummy word to the data FIFO with the End-Of-Event bit set, and send Control EOE to OPORT to start output.

Example Syntax: MOVE.L DUMMY,END_OF_EVENT

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: set EOE flag to data FIFO; DEOE⇑
set FIFO data write; SDW⇑
C5: negate FIFO data write;
C6: set FIFO data write; SDW⇑;
C7: negate FIFO data write
negate EOE flag to data FIFO
set EOE flag to Output Port; CEOE⇑;
C8: hold EOE flag to Output Port true
return sequencer status;

4.1.26 END_OF_EVENT_REXMIT

END_OF_EVENT_REXMIT	FASTBASE+$02C

Description: Send Control EOE to OPORT to start output. Does not insert the Data EOE flag into the Data FIFO. Use End_Of_Event_Rexmit when using the Retransmit feature of the Data FIFO to repeat an event transfer out of the Output Port. Use End_Of_Event when the data in the Data FIFO has not been output before.

Example Syntax: MOVE.L DUMMY,END_OF_EVENT_REXMIT

Operation: C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {set EOE flag to output controller; CEOE⇑}
C5: return sequencer status;

4.1.27 SLAVE_DATA_INPUT

SLAVE_DATA_INPUT	SLOWBASE+$01C

Description: Transfer one data word from FASTBUS to the processor in slave mode.

Example Syntax: MOVE.L SLAVE_DATA_INPUT,DATA

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTS;
      elseif  WT(FRWT)    goto C4;
      elseif  DS(FRDS)    {return processor acknowledge A; FDSACK*⇓
                           clear WT;} FCWT⇑
      else                {return processor acknowledge A; FDSACK*⇓
                           clear WT; FCWT⇑
                           goto C6;}
C5:   {return processor acknowledge B; FDSACK*⇓
      DK=1; FSDK⇑
      goto C7;}
C6:   {return processor acknowledge B; FDSACK*⇓
      DK=0;} FCDK⇑
C7:  return sequencer status;
C8:  delay cycle;
C9:  delay cycle;
```

4.1.28 SLAVE_DATA_OUTPUT

SLAVE_DATA_OUTPUT	FASTBASE+$344

Description: Transfer one data word from the processor to FASTBUS in slave mode.

Example Syntax: MOVE.L DATA,SLAVE_DATA_OUTPUT

Operation:

```
C1:   FPORT select;
      latch DATA; DCPBA⇑
      return processor acknowledge A; FDSACK*⇓
C2:   FPORT instruction fetch;
      return processor acknowledge B; FDSACK*⇓
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT) goto INTF;
      elseif  WT(FRWT)    got0 C4;
      elseif  DS(FRDS)    {clear WT; FCWT⇑
                           DK=1; FSDK⇑
                           return sequencer status;
                           exit;}
      else                {clear WT; FCWT⇑
                           DK=0; FCDK⇑
                           return sequencer status;
                           exit;}
```

PIPELINE_TERMINATE — Internal Subroutine

Description: Internal routine to terminate pipelined transfers.

Example Syntax:

Operation:

```
C1:     if    SS1(FRSS1) clock global work counter; FCLK⇑
C2:     if      IRQ(SEQINT) goto INTF;
        elseif  DS(FRDS)    continue; TIMER⇑
        else            goto C8;
/* termination routine for odd word count transfer */
/* check that DK is high */
/* set DS low and wait for DK low */
C3:     if      IRQ(SEQINT) goto INTF;
        else    ! DK(! FRDK) goto C3; TIMER⇑
C4:     continue;                   /* delay cycle */
C5:     continue;                   /* delay cycle */
C6:     if      IRQ(SEQINT) goto INTF;
        elseif  WT(FRWT)    goto C6;
        else            {clock global word counter; FCLK⇑
                         DS=0;} FCDS⇑
C7:     if      IRQ(SEQINT) goto INTF;
        elseif  ! DK(! FRDK) {RD=0; FDRD⇓
                        return sequencer status
                            exit;}
        else            goto C7; TIMER⇑
/* termination routine for even word count transfer */
/* check that DK is low */
C8:if   IRQ(SEQINT)     goto INTF;
        elseif  ! DK(! FRDK) {reset short timer; TIMER⇑
                        return sequencer status}
        else            goto C8;
C9:continue;
C10:continue;
C11:exit;
```

INTS — Internal Subroutine

Description: Internal routine to abort instruction on error interrupt.
INTS assumes that processor acknowledge has not yet been returned.

Operation:
C1: return processor acknowledge A; FDSACK*⇓
C2: return processor acknowledge B; FDSACK*⇓
goto INTF;

INTF — Internal Subroutine

Description: Internal routine to abort instruction on error interrupt.
INTF assumes that processor acknowledge has already been returned.
Exit the current FASTBUS operation by returning all signals to inactive state.

Operation:

```
C1:   {disable transceivers;
      AS=0; FCAS⇑
      DS=0; FCDS⇑
      DK=0;} FCDK⇑
C2:   delay cycle;      /* processor deselect */
C3:   exit;
```

4.2 FPORT Controller List Mode Instruction Set

4.2.1 BUS_ARBITRATE

BUS_ARBITRATE	LISTBASE+$300

Description: Arbitrate for FASTBUS using the low byte of the data operand. Bits 0-5 supply the arbitration vector. Bit 7 enables assured access mode. Bit 6 (prioritized access mode) is ignored. Note that the data operand is a long word and is normally identical to the value of CSR 8.

Example Syntax: MOVE.L CSR_8, L_BUS_ARBITRATE

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: if IRQ(SEQINT) {goto INTS;}
elseif IGK(FRDY) return sequencer status; return
else {request bus; FREQ⇑ goto C4;}

4.2.2 BUS_RELEASE

BUS_RELEASE	LISTBASE+$004

Description: Release FASTBUS arbitration lock.

Example Syntax: MOVE.L DUMMY, L_BUS_RELEASE

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {DS=0; FCDS⇑
DK=0; FCDK⇑
AS=0; FCAS⇑
release bus;} FREL⇑
C5: return sequencer status;
C6: Delay Cycle; /*FPORT deselect*/
C7: Delay Cycle;
C8: Delay Cycle;

4.2.3 ADDRESS_DATA_GEOGRAPHICAL

ADDRESS_DATA_GEOGRAPHICAL	LISTBASE+$304

Description: Perform a FASTBUS geographical primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR, L_ADDRESS_DATA_GEOGRAPHICAL

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  goto C4;}
        elseif      AK(FRAK)      {reset short timer; TIMER⇓
                                  MS=0;
                                  return sequencer status;
                                  exit;}
        else                      {AS=1; FSAS⇑
                                  RD=0;
                                  MS=0;
                                  EG=1; FDEG⇑
                                  enable short timer;
                                  goto C4;} TIMER⇑
```

4.2.4 ADDRESS_CSR_GEOGRAPHICAL

ADDRESS_CSR_GEOGRAPHICAL	LISTBASE+$308

Description: Perform a FASTBUS geographical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,L_ADDRESS_CSR_GEOGRAPHICAL

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  goto C4;}
        elseif      AK(FRAK)      {reset short timer; TIMER⇓
                                  MS=0; FDMS0⇓
                                  return sequencer status;
                                  exit;}
        else                      {AS=1; FSAS⇑
                                  RD=0;
                                  MS=1; FDMS0⇑
                                  EG=1; FDEG⇑
                                  enable short timer; TIMER⇑
                                  goto C4;}
```

4.2.5 ADDRESS_DATA_LOGICAL

ADDRESS_DATA_LOGICAL	LISTBASE+$30C

Description: Perform a FASTBUS logical primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR,L_ADDRESS_DATA_LOGICAL

Operation:

```
C1:   FPORT select;
      latch FB_ADDR; DCPBA⇑
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if          IRQ(SEQINT)   goto INTF;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                goto C4;}
      elseif      AK(FRAK)      {reset short timer; TIMER⇓
                                MS=0;
                                return sequencer status;
                                exit;}
      else                      {AS=1; FSAS⇑
                                RD=0;
                                enable short timer; TIMER⇑
                                goto C4;}
```

4.2.6 ADDRESS_CSR_LOGICAL

ADDRESS_CSR_LOGICAL	LISTBASE+$310

Description: Perform a FASTBUS logical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,L_ADDRESS_CSR_LOGICAL

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                    goto C4;}
        elseif  AK(FRAK)    {reset short timer; TIMER⇓
                    MS=0; FDMS0⇓
                    return sequencer status;
                    exit;}
        else            {AS=1; FSAS⇑
                    RD=0;
                    MS=1; FDMS0⇑
                    enable short timer; TIMER⇑
                    goto C4;}
```

4.2.7 ADDRESS_DATA_BROADCAST

ADDRESS_DATA_BROADCAST	LISTBASE+$314

Description: Perform a FASTBUS broadcast primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR,L_ADDRESS_DATA_BROADCAST

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  goto C4;}
        elseif      AK(FRAK)      {reset short timer; TIMER⇓
                                  MS=0; FDMS1⇓
                                  return sequencer status;
                                  exit;}
        else                      {AS=1; FSAS⇑
                                  RD=0;
                                  MS=2; FDMS1⇑
                                  enable short timer;} TIMER⇑
```

4.2.8 ADDRESS_CSR_BROADCAST

ADDRESS_CSR_BROADCAST	LISTBASE+$318

Description: Perform a FASTBUS geographical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,L_ADDRESS_CSR_BROADCAST

Operation:

```
C1:     FPORT select;
        latch FB_ADDR; DCPBA⇑
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  goto C4;}
        elseif      AK(FRAK)      {reset short timer; TIMER⇓
                                  MS=0; FDMS0⇓, FDMS1⇓
                                  return sequencer status;
                                  exit;}
        else                      {AS=1; FSAS⇑
                                  RD=0;
                                  MS=3; FDMS0⇑, FDMS1⇑
                                  enable short timer; TIMER⇑
                                  goto C4;}
```

4.2.9 ADDRESS_RELEASE

ADDRESS_RELEASE	LISTBASE+$31C

Description: Release address lock.

Example Syntax: MOVE.L DUMMY,L_ADDRESS_RELEASE

Operation:

```
C1:     FPORT select;
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  goto C4;}
        else                      {DS=0; FCDS⇑
                                  DK=0; FCDK⇑
                                  MS=0; FDMS0⇓, FDMS1⇓,FDMS2⇓
                                  AS=0; FSAS⇑
                                  enable short timer;} TIMER⇑

C5: if          IRQ(SEQINT)  goto INTF;
        elseif  AK(FRAK)     goto C5;
        else                 return sequencer status;
                             exit;
```

4.2.10 DATA _RANDOM_READ

DATA _RANDOM_READ	LISTBASE+$320

Description: Perform a FASTBUS single word read data cycle. FASTBUS Data word is transferred to the Data FIFO.

Example Syntax: MOVE.L DUMMY,L_DATA _RANDOM_READ

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if          IRQ(SEQINT)   goto INTS;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                  goto C4;}
      elseif      DK(FRDK)
      else        {enable short timer; TIMER⇑
                  RD=1; FDRD⇑
                  MS=0;
                  DS=1; FSDS⇑
                  goto C4;}
C5:   reset short timer; TIMER⇓
C6:   if          IRQ(SEQINT)   INTF;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                  goto C6;}
      elseif      !DK(FRDK*)    {return sequencer status;
                  reset short timer; TIMER⇓}
      else        {enable short timer; TIMER⇑
                  RD=0; FDRD⇓
                  MS=0;
                  DS=0; FCDS⇑
                  goto C6;}
C7:   delay cycle;
C8:   delay cycle;
C9:   delay cycle;
```

4.2.11 DATA _RANDOM_WRITE

DATA _RANDOM_WRITE	LISTBASE+$324

Description: Perform a FASTBUS single word write data cycle.

Example Syntax: MOVE.L DATA,L_DATA _RANDOM_WRITE

Operation:

```
C1:   FPORT select;
      latch DATA; DCPBA⇑
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if          IRQ(SEQINT)   goto INTF;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                  goto C4;}
      elseif      DK(FRDK)      reset short timer; TIMER⇓
      else        {enable short timer; TIMER⇑
                  RD=0;
                  MS=0;
                  DS=1; FSDS⇑
                  goto C4;}
C5:   if          IRQ(SEQINT)   goto INTF;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                  goto C5;}
      elseif      !DK(FRDK*)    {return sequencer status;
                  exit;}
      else        {enable short timer; TIMER⇑
                  RD=0;
                  MS=0;
                  DS=0; FCDS⇑
                  goto C5;}
```

4.2.12 DATA _SECONDARY_ADDRESS_READ

DATA _SECONDARY_ADDRESS_READ	LISTBASE+$328

Description: Perform a FASTBUS secondary address read cycle. Data is transferred to the Data FIFO.

Example Syntax: MOVE.L DUMMY,L_DATA _SECONDARY_ADDRESS_READ

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if          IRQ(SEQINT)   goto INTS;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                  goto C4;}
      elseif      DK(FRDK)
      else        {enable short timer; TIMER⇑
                  RD=1; FDRD⇑
                  MS=2; FDMS1⇑
                  DS=1; FSDS⇑
                  goto C4;}
C5:   reset short timer; TIMER⇓
C6:   if          IRQ(SEQINT)   goto INTF;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                  goto C6;}
      elseif      !DK(FRDK*)    {reset short timer; TIMER⇓
                  return sequencer status;}
      else        {enable short timer; TIMER⇑
                  RD=0; FDRD⇓
                  MS=2; FDMS1⇑
                  DS=0; FCDS⇑
                  goto C6;}
C7:   delay cycle;      /* processor deselect */
C8:   delay cycle;
C9:   delay cycle;
```

4.2.13 DATA _SECONDARY_ADDRESS_WRITE

DATA _SECONDARY_ADDRESS_WRITE	LISTBASE+$32C

Description: Perform a FASTBUS secondary address write cycle.

Example Syntax: MOVE.L SADDR,L_DATA _SEC_ADDRESS_WRITE

Operation:

```
C1:     FPORT select;
        latch SADDR; DCPBA⇑
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  goto C4;}
        elseif      DK(FRDK)      reset short timer; TIMER⇓
        else                      {enable short timer; TIMER⇑
                                  RD=0;
                                  MS=2; FDMS1⇑
                                  DS=1; FSDS⇑
                                  goto C4;}
C5:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  goto C5;}
        elseif      !DK(FRDK*)    {return sequencer status;
                                  exit;}
        else                      {enable short timer; TIMER⇑
                                  RD=0;
                                  MS=0; FDMS1⇓
                                  DS=0; FCDS⇑
                                  goto C5;}
```

4.2.14 DATA _BLOCK_TRANSFER_WRITE

DATA _BLOCK_TRANSFER_WRITE

LISTBASE+$00C

Description: Output one word of a FASTBUS block transfer write cycle. Since the FSCC has been optimized as a read-out controller, there is no FIFO to queue data to be output through the FASTBUS port. Block transfer writes occur by executing one DATA_BLOCK_TRANSFER_WRITE instruction for each data word to be output. The FASTBUS sequencer maintains the proper state of DS after completion of the instruction. The DATA_BLOCK_TRANSFER_TERMINATE instruction must then be used to "clean-up" after a string of DATA_BLOCK_TRANSFER_WRITE instructions have been executed. Note that the state of the MS lines are set to zero after each DATA_BLOCK_TRANSFER_WRITE instruction is executed. This is allowed by the FASTBUS specification, however, some slaves are confused by this behavior. Since the DATA_BLOCK_TRANSFER_WRITE instruction effectively leaves the state of the Sequencer, and of the attatched slave in the middle of a block transfer operation, care should be taken to ensure that a DATA_BLOCK_TRANSFER_TERMINATE instruction always follows any group of DATA_BLOCK_TRANSFER_WRITE instructions.

Example Syntax: MOVE.L DATA,L_DATA _BLOCK_TRANSFER_WRITE

Operation:

```
C1:   FPORT select;
      latch DATA; DCPBA⇑
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT)  goto INTF;
      elseif  WT(FRWT)     {reset short timer; TIMER⇓
                  RD=0;
                  MS=1; FDMS0⇑
                  goto C4;}
      elseif  ! DS(FRDS*)  {enable short timer; TIMER⇑
                  RD=0;
                  MS=1; FDMS0⇑
                  DS=1; FSDS⇑
                  goto C6;}
      else                 {enable short timer; TIMER⇑
                  RD=0;
                  MS=1; FDMS0⇑
                  DS=0;} FCDS⇑
C5:   if      IRQ(SEQINT)  goto INTF;
      elseif  ! DK(FRDK*)  {reset short timer; TIMER⇓
                  RD=0;
                  MS=0; FDMS0⇓
                  return sequencer status;
                  exit;}
      else                 goto C5;
C6:   if      (IRQ)        goto INTF;
```

```
elseif    (DK)      {reset short timer; TIMER⇓
                    RD=0;
                    MS=0; FDMS0⇓
                    exit;}
else                      goto C6;
```

4.2.15 DATA _BLOCK_TRANSFER_TERMINATE

DATA _BLOCK_TRANSFER_TERMINATE	LISTBASE+$330

Description: Perform termination step of a FASTBUS block transfer write operation. This instruction allows the state of DS to be set low after executing one or more DATA_BLOCK_TRANSFER_WRITE instructions. It should always be executed after a group of DATA_BLOCK_TRANSFER_WRITE instructions have been executed.

Example Syntax: MOVE.L DUMMY,L_DATA _BLOCK_TRANSFER_TERMINATE

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if      IRQ(SEQINT)   goto INTF;
      elseif  WT(FRWT)      goto C4;
      elseif  !FRDK         {return sequencer status; goto C5;}
      else                  {MS=0; FDMS0⇓, FDMS1⇓, FDMS2⇓
                             RD=0; FDRD⇓
                             DS=0;} FCDS⇑
C5:   delay cycle;
C6:   delay cycle;
C7:   delay cycle;
```

4.2.16 DATA _BLOCK_TRANSFER_READ

DATA _BLOCK_TRANSFER_READ	LISTBASE+$334

Description: Perform a FASTBUS block transfer read to the Data FIFO. The block transfer is terminated upon receipt of SS=2 from the slave, or if the number of words specified in the Local Counter have been read (if enabled).

Example Syntax: MOVE.L DUMMY,L_DATA _BLOCK_TRANSFER_READ

Operation:

```
C1:     FPORT select;
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                        enable FIFO; DFIFOEN⇑,PFIFOEN⇑
                        RD=1; FDRD⇑
                        MS=1; FDMS0⇑
                        goto C4;}
        else            {enable FIFO; DFIFOEN⇑,PFIFOEN⇑
                        RD=1; FDRD⇑
                        MS=1; FDMS0⇑
                        DS=1; FSDS⇑
                        local counter mode = decrement; LC0⇑, LC1⇑}
C5:     {local counter mode = hold; LC0⇓, LC1⇓
        enable short timer;} TIMER⇑
C6:     continue;               /* delay cycle */
C7:     if          ((FRDK*FRSS1)#SEQINT #FEOB)   goto C11;
        elseif  WT(FRWT)    {reset short timer; TIMER⇓
                        goto C7;}
        elseif  DK(FRDK)    {DS=0; FCDS⇑
                        local counter mode = decrement; LC0⇑, LC1⇑
                        reset short timer; TIMER⇓
                        clock global word counter; FCLK⇑}
        else            {enable short timer; TIMER⇑
                        goto C7;}
C8:     {local counter mode = hold; LC0⇓, LC1⇓    /* delay cycle */
                        enable short timer; TIMER⇑}
C9: delay cycle;
C10: if         ((!FRDK*FRSS1)#SEQINT#FEOB)  goto C11;
        elseif  WT(FRWT)    {reset short timer;
                        goto C10;}
        elseif  !DK(!FRDK)  {DS=1; FSDS⇑
                        local counter mode=decrement; LC0⇑, LC1⇑
                        reset short timer; TIMER⇓
                        clock global word counter; FCLK⇑
                        goto C5;}
        else            goto C10;
```

```
C11: if          SS2(FRSS1)    clock global word counter; FCLK⇑
/* block transfer termination */
C12: if          IRQ(SEQINT)   goto INTF;
     elseif      DS(FRDS)      continue;
     else                      goto C22;
/* termination routine for odd word count transfer */
/* check that DK is high */
/* set DS low and wait for DK low */
/* data written to FIFO on DK down is dummy word with EOE flag*/
C13:if           IRQ(SEQINT)   goto INTF;
    elseif       DK(FRDK)      {reset short timer; TIMER⇓
                               MS=0;} FDMS0⇓
    else                       {enable short timer; TIMER⇑
                               goto C13;}
C14:continue;                  /* delay cycle */
C15:continue;                  /* delay cycle */
C16:continue;                  /* delay cycle */
C17:if           IRQ(SEQINT)   goto INTF;
    elseif       WT(FRWT)      {reset short timer; TIMER⇓
                               goto C17;}
    else                       {enable short timer; TIMER⇑
                               DS=0;} FCDS⇑
C18:if           IRQ(SEQINT)   goto INTF;
    elseif       ! DK(! FRDK)  {RD=0; FDRD⇓
                               reset short timer; TIMER⇓
                               return sequencer status}
    else                       {enable short timer; TIMER⇑
                               goto C18;}
C19:continue;                  /* delay cycle */
C20:continue;                  /* delay cycle */
C21:exit;
/* termination routine for even word count transfer */
C22:if           IRQ(SEQINT)   goto INTF;    /* check that DK is low */
    elseif       ! DK(! FRDK)  {RD=0; FDRD⇓
                               reset short timer; TIMER⇓
                               return sequencer status}
    else                       {enable short timer; TIMER⇑
                               goto C22;}
C23:continue;                  /* delay cycle */
C24:continue;                  /* delay cycle */
C25:continue;                  /* delay cycle */
C25:exit;
```

4.2.17 DATA _PIPELINED_READ_100

DATA _PIPELINED_READ_100	LISTBASE+$338

Description: Perform a FASTBUS pipelined read to the Data FIFO at 100 nsec/word. This instruction functions similarly to the DATA_BLOCK_TRANSFER_READ instruction except that it follows the FASTBUS specification for Pipelined operations. Note that the FSCC does not implement a DK counter as called for in the FASTBUS specification.

Example Syntax: MOVE.L DUMMY,L _PIPELINED_READ_100

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if          IRQ(SEQINT)   goto INTF;
      elseif      WT(FRWT)      goto C4;
      elseif      EOB(FEOB)     goto PIPELINE_TERMINATE;
      else              {enable FIFO; PFFEN⇑, DFFEN⇑
                        RD=1; FDRD⇑
                        MS=3; FDMS0⇑,FDMS1⇑
                        DS=1; FSDS⇑
                        local counter mode = decrement; LC0⇑,LC1⇑
                        clock global word counter;} FCLK⇑
C5:   local counter mode = hold; LC0⇓,LC1⇓
C6:   if          IRQ(SEQINT)   goto INTF;
      elseif      WT(FRWT)      goto C6;
      elseif      EOB(FEOB)     goto PIPELINE_TERMINATE;
      else              {DS=0; FCDS⇑
                        local counter mode = decrement; LC0⇑,LC1⇑
                        clock global word counter;} FCLK⇑
C7:   {local counter mode = hold; LC0⇓,LC1⇓
      goto C4;}
```

4.2.18 DATA _PIPELINED_READ_200

DATA _PIPELINED_READ_200	LISTBASE+$33C

Description: Perform a FASTBUS pipelined read to the Data FIFO at 200 nsec/word. This instruction functions similarly to the DATA_BLOCK_TRANSFER_READ instruction except that it follows the FASTBUS specification for Pipelined operations. Note that the FSCC does not implement a DK counter as called for in the FASTBUS specification.

Example Syntax: MOVE.L DUMMY,L_DATA _PIPELINED_READ_200

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if        IRQ(SEQINT)   goto INTF;
      elseif    WT(FRWT)      goto C4;
      elseif    EOB(FEOB)     goto PIPELINE_TERMINATE;
      else      {enable FIFO; PFFEN⇑, DFFEN⇑
                 RD=1; FDRD⇑
                 MS=3; FDMS0⇑,FDMS1⇑
                 DS=1; FSDS⇑
                 local counter mode = decrement; LC0⇑,LC1⇑
                 clock global word counter;} FCLK⇑
C5:   local counter mode = hold; LC0⇓,LC1⇓
C6:   continue;           /* delay cycle */
C7:   continue;           /* delay cycle */
C8:   if        IRQ(SEQINT)   goto INTF;
      elseif    WT(FRWT)      goto PIPELINE_TERMINATE;
      elseif    EOB(FEOB)     goto C12;
      else      {DS=0; FCDS⇑
                 local counter mode = decrement; LC0⇑,LC1⇑
                 clock global word counter;} FCLK⇑
C9:   local counter mode = hold; LC0⇓,LC1⇓
C10:continue;             /* delay cycle */
C11:goto C4;              /* delay cycle */
```

4.2.19 DATA _PIPELINED_READ_400

DATA _PIPELINED_READ_400	LISTBASE+$340

Description: Perform a FASTBUS pipelined read to the Data FIFO at 400 nsec/word. This instruction functions similarly to the DATA_BLOCK_TRANSFER_READ instruction except that it follows the FASTBUS specification for Pipelined operations. Note that the FSCC does not implement a DK counter as called for in the FASTBUS specification.

Example Syntax: MOVE.L DUMMY,L_DATA _PIPELINED_READ_400

Operation:

```
C1:     FPORT select;
        return processor acknowledge A; FDSACK*⇓
C2:     FPORT instruction fetch;
        return processor acknowledge B; FDSACK*⇓
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      goto C4;
        elseif      EOB(FEOB)     goto PIPELINE_TERMINATE;
        else                {enable FIFO; PFFEN⇑, DFFEN⇑
                            RD=1; FDRD⇑
                            MS=3; FDMS0⇑,FDMS1⇑
                            DS=1; FSDS⇑
                            local counter mode = decrement; LC0⇑,LC1⇑
                            clock global word counter;} FCLK⇑
C5:     local counter mode = hold; LC0⇓,LC1⇓
C6:     continue;                   /* delay cycle */
C7:     continue;                   /* delay cycle */
C8:     continue;                   /* delay cycle */
C9:     continue;                   /* delay cycle */
C10:continue;               /* delay cycle */
C11:continue;               /* delay cycle */
C12:if          IRQ(SEQINT) goto INTF;
        elseif      WT(FRWT)      goto C12;
        elseif      EOB(FEOB)     goto PIPELINE_TERMINATE;
        else                {DS=0; FCDS⇑
                            local counter mode = decrement; LC0⇑,LC1⇑
                            clock global word counter;} FCLK⇑
C13:local counter mode = hold; LC0⇓,LC1⇓
C14:continue;               /* delay cycle */
C15:continue;               /* delay cycle */
C16:continue;               /* delay cycle */
C17:continue;               /* delay cycle */
C18:continue;               /* delay cycle */
C19:goto C4;                /* delay cycle */
```

4.2.20 DATA _RANDOM_READ_LEADING_WORD_COUNT

DATA _RANDOM_READ_LEADING_WORD_COUNT LISTBASE+$344

Description: Perform a FASTBUS single word read data cycle. FASTBUS Data word is transferred to the Local Word Counter (LWC).

Example Syntax: MOVE.L DUMMY,L_DATA _RANDOM_READ_LEAD_LEADING_WORD_COUNT

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   if          IRQ(SEQINT)   goto INTF;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                  goto C4;}
      elseif      DK(FRDK)      {counter load, goto C5;}
      else        {enable short timer; TIMER⇑
                  RD=1; FDRD⇑
                  MS=0;
                  DS=1; FSDS⇑
                  goto C4;}
C5:   {counter load, reset short timer; TIMER⇓}
C6:   if          IRQ(SEQINT)   INTF;
      elseif      WT(FRWT)      {reset short timer; TIMER⇓
                  goto C6;}
      elseif      !DK(FRDK*)    {return sequencer status;
                  reset short timer; TIMER⇓}
      else        {enable short timer; TIMER⇑
                  RD=0; FDRD⇓
                  MS=0;
                  DS=0; FCDS⇑
                  goto C6;}
C7:   delay cycle;
C8: delay cycle;
C9: delay cycle;
```

4.2.21 DATA _BLOCK_TRANSFER_READ_TO_LOCAL_COUNTER

DATA _BLOCK_TRANSFER_READ_LOCAL_COUNTER
LISTBASE+$348

Description: Perform a FASTBUS block transfer read to the Data FIFO, place the first word transferred into the Local counter. If the Local Counter is enabled, the transfer will then terminate after the number of words specifed in the first word of the transfer (non-inclusive) have been input.

Example Syntax: MOVE.L DUMMY,L_DATA _BLOCK_TRANSFER_READ_TO...

Operation:

```
C1:     FPORT select;
C2:     FPORT instruction fetch;
C3:     instruction dispatch;
C4:     if          IRQ(SEQINT)   goto INTF;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  enable FIFO; DFIFOEN⇑,PFIFOEN⇑
                                  RD=1; FDRD⇑
                                  MS=1; FDMS0⇑
                                  goto C4;}
        elseif      DK(FRDK)      local counter mode = load; LC0⇓, LC1⇑
                                  goto C5;}
        else                      {RD=1; FDRD⇑
                                  MS=1; FDMS0⇑
                                  DS=1; FSDS⇑
                                  goto C4;}
C5:     {local counter mode = load; LC0⇓, LC1⇑
        goto }
C6:     continue;                 /* delay cycle */
C7:     if          ((FRDK*FRSS1)#SEQINT #FEOB)    goto C11;
        elseif      WT(FRWT)      {reset short timer; TIMER⇓
                                  goto C7;}
        elseif      DK(FRDK)      {DS=0; FCDS⇑
                                  local counter mode = decrement; LC0⇑, LC1⇑
                                  reset short timer; TIMER⇓
                                  clock global word counter; FCLK⇑}
        else                      {enable short timer; TIMER⇑
                                  goto C7;}
C8:     {local counter mode = hold; LC0⇓, LC1⇓      /* delay cycle */
                                  enable short timer; TIMER⇑}
C9: delay cycle;
C10: if         ((!FRDK*FRSS1)#SEQINT#FEOB)  goto C11;
        elseif      WT(FRWT)      {reset short timer;
                                  goto C10;}
        elseif      !DK(!FRDK)    {DS=1; FSDS⇑
                                  local counter mode=decrement; LC0⇑, LC1⇑
                                  reset short timer; TIMER⇓
```

```
                                clock global word counter; FCLK⇑
                                goto C5;}
          else                  goto C10;
C11: if             SS2(FRSS1)  clock global word counter; FCLK⇑
/* block transfer termination */
C12: if             IRQ(SEQINT) goto INTF;
           elseif   DS(FRDS)    continue;
           else                 goto C22;
/* termination routine for odd word count transfer */
/* check that DK is high */
/* set DS low and wait for DK low */
/* data written to FIFO on DK down is dummy word with EOE flag*/
C13:if              IRQ(SEQINT) goto INTF;
          elseif    DK(FRDK)    {reset short timer; TIMER⇓
                                MS=0;} FDMS0⇓
          else                  {enable short timer; TIMER⇑
                                goto C13;}
C14:continue;                       /* delay cycle */
C15:continue;                       /* delay cycle */
C16:continue;                       /* delay cycle */
C17:if              IRQ(SEQINT) goto INTF;
          elseif    WT(FRWT)    {reset short timer; TIMER⇓
                                goto C17;}
          else                  {enable short timer; TIMER⇑
                                DS=0;} FCDS⇑
C18:if              IRQ(SEQINT) goto INTF;
          elseif    ! DK(! FRDK) {RD=0; FDRD⇓
                                reset short timer; TIMER⇓
                                return sequencer status}
          else                  {enable short timer; TIMER⇑
                                goto C18;}
C19:continue;                       /* delay cycle */
C20:continue;                       /* delay cycle */
C21:exit;
/* termination routine for even word count transfer */
C22:if              IRQ(SEQINT) goto INTF;    /* check that DK is low */
          elseif    ! DK(! FRDK) {RD=0; FDRD⇓
                                reset short timer; TIMER⇓
                                return sequencer status}
          else                  {enable short timer; TIMER⇑
                                goto C22;}
C23:continue;                       /* delay cycle */
C24:continue;                       /* delay cycle */
C25:continue;                       /* delay cycle */
C25:exit;
```

4.2.22 TRIGGER_HOLD

TRIGGER_HOLD	LISTBASE+$02C

Description: Poll Trigger FIFO empty flag and wait if FIFO is empty.

Example Syntax: MOVE.L DUMMY,L_TRIGGER_HOLD

Operation:
```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   delay;
C5:   if          IRQ(SEQINT)   goto INTF;
      elseif      TFIFO_NOT_EMPTY     {exit;}
      else              {goto C4;}
```

4.2.23 INSTRUCTION_LIST_RE-EXECUTE

INSTRUCTION_LIST_RE-EXECUTE	LISTBASE+$030

Description: Toggle List FIFO Retransmit Input.

Example Syntax: MOVE.L DUMMY,L_INSTRUCTION_LIST_RE-EXECUTE

Operation: C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {set list FIFO re-ex input true, SRT⇑;}
C5: {set list FIFO re-ex input true, SRT⇑;}
C6: {set list FIFO re-ex input true, SRT⇑;}
C7: {set list FIFO re-ex input true, SRT⇑
exit;}

4.2.24 GENERATE_FPCREQ (IRQ)

GENERATE_FPCREQ (IRQ)	LISTBASE+$01C

Description: Toggle FPCREQ interrupt to CPU.

Example Syntax: MOVE.L DUMMY,L_GENERATE_FPCREQ

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {set FPCREQ interrupt line true, FPCREQ⇑;}
C5: {set FPCREQ interrupt line true, FPCREQ⇑;
exit;}

4.2.25 POLL_HALT_REQUEST

POLL_HALT_REQUEST	LISTBASE+$034

Description: Check for HALT_REQUEST from CPU. Halt Sequencer if true, exit if false.

Example Syntax: MOVE.L DUMMY,L_POLL_HALT_REQUEST

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: delay;
C5: if HALT_REQUEST {return processor acknowledge A, FDSACK*⇓,goto C6;}
else exit;
C6: {return processor acknowledge B, FDSACK*⇓}
C7: goto C7 /*stay here forever to "HALT" the sequencer*/

4.2.26 DELAY2

DELAY2	LISTBASE+$038

Description: Cause a 2 microsecond delay between previous instruction completion, and following instruction initiation.

Example Syntax: MOVE.L DUMMY,L_DELAY2

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: load Sequencer counter with 31d;
C5: decrement Sequencer counter and loop to C5 until counter = 0;
C6: exit;

4.2.27 DELAY10

DELAY10	LISTBASE+$03C

Description: Cause a 10 microsecond delay between previous instruction completion, and following instruction initiation.

Example Syntax: MOVE.L DUMMY,L_DELAY10

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: load Sequencer counter with 191d;
C5: decrement Sequencer counter and loop to C5 until counter = 0;
C6: exit;

4.2.28 DELAY100

DELAY100	LISTBASE+$040

Description: Cause a 100 microsecond delay between previous instruction completion, and following instruction initiation.

Example Syntax: MOVE.L DUMMY,L_DELAY2

Operation: C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: load Sequencer counter with 10d;
C5: push Sequencer counter value into stack and load counter with 177d;
C6: decrement Sequencer counter and loop to C6 until counter = 0;
C7: load Sequencer counter with value popped from stack;
C8: decrement Sequencer counter and loop to C5 until counter = 0;
C9: delay
C10: exit;

4.2.29 SEQUENCER_NULL

SEQUENCER_NULL	LISTBASE+$020

Description: Access the FPORT Controller without performing any operation.

Example Syntax: MOVE.L DUMMY,L_SEQUENCER_NULL

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {return sequencer status;
exit;}

4.2.30 BULB_TEST

BULB_TEST	LISTBASE+$028

Description: Diagnostic to set all driven FASTBUS lines true.

Example Syntax: MOVE.L DUMMY,L_BULB_TEST

Operation: C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {Set FPCREQ interrupt; FPCREQ⇑,
latch output data,
EG=1; FDEG⇑,
RD=1; FDRD⇑,
MS=7; FDMS0⇑,FDMS1⇑,FDMS2⇑,
DK=1; FSDK⇑,
DS=1; FSDS⇑,
AS=1; FSAS,⇑
goto C4}

4.2.31 LOCAL_COUNTER_LOAD

LOCAL_COUNTER_LOAD	LISTBASE+$010

Description: Initialize Local Word Counter for block transfer.

Example Syntax: MOVE.L COUNT,L_LOCAL_COUNTER_LOAD

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {local counter mode = load;} LC1⇑
C5: {local counter mode = hold;} LC1⇓
C6: return sequencer status;
C7: delay cycle;
C8: delay cycle

Note: The counter is loaded from the low order 12 bits of COUNT.

4.2.32 LOCAL_COUNTER_READ

LOCAL_COUNTER_READ	SLOWBASE+$014

Description: Transfers current value of Local Word Counter into Data FIFO.

Example Syntax: MOVE.L DUMMY,L_LOCAL_COUNTER_READ

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {local counter mode = read;} SLCOE⇑
C5: return sequencer status;
C6: delay cycle;
C7: delay cycle;

4.2.33 FIFO_WRITE_DATA

FIFO_WRITE_DATA	LISTBASE+$018

Description: Write a single word from the processor to the Data FIFO.

Example Syntax: MOVE.L DATA,L_FIFO_WRITE_DATA

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {clock FIFO input;} SDW⇑
C5: {clock global word counter;} FCLK⇑
C6: return sequencer status;
C7: delay cycle;
C8: delay cycle;

4.2.34 END_OF_EVENT

END_OF_EVENT	LISTBASE+$024

Description: Write dummy word to the data FIFO with the End-Of-Event bit set, and send Control EOE to OPORT to start output.

Example Syntax: MOVE.L DUMMY,L_END_OF_EVENT

Operation:
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: set EOE flag to data FIFO; DEOE⇑
set FIFO data write; SDW⇑
C5: negate FIFO data write;
C6: set FIFO data write; SDW⇑;
C7: negate FIFO data write
negate EOE flag to data FIFO
set EOE flag to Output Port; CEOE⇑;
C8: hold EOE flag to Output Port true
return sequencer status;

4.2.35 TRIGGER_HOLD_WITH_HALT_REQUEST

TRIGGER_HOLD_WITH_HALT_REQUEST	LISTBASE+$02C

Description: Poll Trigger FIFO empty flag and Halt Request from CPU. If TFIFO is not empty, then exit, elseif HALT_REQUEST, then halt, else wait.

Example Syntax: MOVE.L DUMMY,L_TRIGGER_STROBE_HOLD_WITH_HALT_REQUEST

Operation:

```
C1:   FPORT select;
C2:   FPORT instruction fetch;
C3:   instruction dispatch;
C4:   delay;
C5:   if      IRQ(SEQINT)          goto INTF;
      elseif  TFIFO_NOT_EMPTY      {exit;}
      elseif  HALT_REQUEST         {return processor acknowledge A, FDSACK*⇓,goto C6;}
      else                         {goto C5;}
C6:   {return processor acknowledge B, FDSACK*⇓}
C7:   goto C7     /*stay here forever to "HALT" the sequencer*/
```

PIPELINE_TERMINATE — Internal Subroutine

Description: Internal routine to terminate pipelined transfers.

Example Syntax:

Operation:

```
C1:     if      SS1(FRSS1)      clock global work counter; FCLK⇑
C2:     if      IRQ(SEQINT)     goto INTF;
        elseif  DS(FRDS)        continue; TIMER⇑
        else                    goto C8;
/* termination routine for odd word count transfer */
/* check that DK is high */
/* set DS low and wait for DK low */
C3:     if      IRQ(SEQINT)     goto INTF;
        else    ! DK(! FRDK)    goto C3; TIMER⇑
C4:     continue;                   /* delay cycle */
C5:     continue;                   /* delay cycle */
C6:     if          IRQ(SEQINT)     goto INTF;
        elseif  WT(FRWT)        goto C6;
        else                {clock global word counter; FCLK⇑
                            DS=0;} FCDS⇑
C7:     if          IRQ(SEQINT)     goto INTF;
        elseif  ! DK(! FRDK)    {RD=0; FDRD⇓
                            return sequencer status
                                exit;}
        else                goto C7; TIMER⇑
/* termination routine for even word count transfer */
/* check that DK is low */
C8:if           IRQ(SEQINT)     goto INTF;
        elseif  ! DK(! FRDK)    {reset short timer; TIMER⇑
                            return sequencer status}
        else                goto C8;
C9:continue;
C10:continue;
C11:exit;
```

INTF — Internal Subroutine

Description: Internal routine to abort instruction on error interrupt.
INTF assumes that processor acknowledge has already been returned.
Exit the current FASTBUS operation by returning all signals to inactive state.

Operation:

```
C1:   {disable transceivers;
      AS=0; FCAS⇑
      DS=0; FCDS⇑
      DK=0;} FCDK⇑
C2:   delay cycle;      /* processor deselect */
C3:   exit;
```

5. Appendix B - FSCC Parts List

Item	Quan	Manf. #	Description	Manf.
1	1	MP037-3.6864 MHz	3.6864 MHz Crystal	CTS
2	3		.01uF Non-Polarized Capacitor	
3	12	8134-HC-6P2	.057" FUSE Sockets	AUGAT
4	5		.1uF 25V Non-Polarized Capacitor	
5	125		.1uF Ceramic, Dip Cap	
6	3	1077-3	PC mount Coaxial Connector, K-Loc	Kings
7	1	1074-1	Panel Mount Coaxial Connector, K-Loc	Kings
8	1	4-2-2	Ground Lug for Panel Mount K-Loc Connector	Kings
7	3	4310R-101-101	10 Pin Sip PAK, 100 Ohms	Bourns
8	2	SL-110-G-19	10 Pin Sip Socket	Samtec
9	2	4310R-102-101	10 Pin Sip, 100 Ohms, 5 Individual Res.	Bourns
10	2	7039-SS MOD E=1/2	10/32 Shoulder Screws	R.A.F. Elect.
11	1		100pF Non-Polarized Capacitor	
12	2	PAL1016P8JC	10KH ECL PAL	TI
13	3	ICO-316-SGG	16 Pin Dip Socket, 300 mils wide	Samtec
14	2	4310R-101-102	1K Ohm, 10-Pin Sip PAK	Bourns
15	1	1N914	1N914 Diode	Motorola
16	1	ICO-320-SGG	20 Pin Dip Socket, 300 mils wide	Samtec
17	1	MP200-20MHz	20.000 MHz Crystal	CTS
18	2		20pF Non-Polarized Capacitor	
19	16	GAL22V10-15LP	22V10 Reprogramable AND-OR Array	Lattice
20	1	GAL20RA10-15LP	20RA10 Reprogramable PAL	Lattice
21	2	EPM5128JC-1	128 macrocell EPLD	Altera
22	19	ICO-324-SGG	24 Pin Dip Socket, 300 mils wide	Samtec
23	7	110-99-328-41-001	28 Pin Dip Socket, 300 mils wide	Preci-Dip
24	15	ICO-628-SGG	28 Pin Dip Socket, 600 mils wide	Samtec
25	8	ICO-432-SGG	32 Pin Dip Socket, 400 mils wide	Samtec
26	8	ICO-632-SGG	32 Pin Dip Socket, 600 mils wide	Samtec
27	2		68 Pin JLCC Socket	AMP
28	1		Single Pin Socket	Samtec
29	1	CPAS-114-ZSGG-13A	114 Pin Grid Array Socket	Samtec
30	2	BBS-132-T-A	Board to Board Stand-off Socket	Samtec
31	1		Board to Board Stand-off Pin	Samtec
32	1		2 x 5 pin, right angle jumper block	Samtec
33	6		2-pin slide on jumper	
34	1	IDT7133S70G	2K X 16-Bit, Dual-Port RAM (Master)	IDT
35	1	IDT7143S70G	2K X 16-Bit, Dual-Port RAM (Slave)	IDT
36	1	MC68020RC20	32-Bit Microprocessor (20 MHz)	Motorola
38	8	CXK581020SP-45	128K X 8-Bit CMOS 45ns Static RAM	Sony
39	2	RA0.304NYL	4-Pin LEMO Connector	LEMO
40	4		4-40 X 1/4" Bind Head Screw	
41	1	MX055GA-2C-40 MHz	40 MHz Oscillator	CTS

42	6	ICO-640-SGG	40 Pin Dip Socket, 600 mils wide	Samtec
43	1	ICA-648-SGG	48 Pin Dip Socket, 600 mils wide	Samtec
44	1	MC10H166	5-Bit Magnitude Comparator	Motorola
45	2		5pF Non-Polarized Capacitor	
46	1	2VP5U9	5V-IN,9V-OUT,DC-DC Converter	Reliability
47	3		6.8uF Polarized Capacitor	
48	8	AM27C512-120	64K X 8 Bit CMOS EPROM 120 ns	AMD
49	4	M27C4001-80XFI	512K X 8 Bit CMOS EPROM 80 ns	SGS Thompson
50	2	MVAS-68-ZSGG-11	68 Pin Grid Array Socket	Samtec
51	1	PT10312	75uH, Pulse Transformer	Datatronics
52	1	N74F823N	9-Bit D-Type Edge-Triggered Flip	Signetics
53	1	558-0202-003	Dialight Green LED with Integral RES	Dialight
54	1	558-0302-003	Dialight Yellow LED with Integral RES	Dialight
55	2	MC10H131	Dual D-Type Master-Slave Flip-Flop	Motorola
56	1	SCN68681C1N40	Duart	Signetics
57	9	BT501KC	ECL/TTL Octal Transceiver/Translator	Brooktree
58	1	P82C501-10MHz	Ethernet Serial Interface	Intel
59	1	DP8392A	Ethernet Transceiver Chip	National
60	2	0882-MB-199070	FASTBUS Front Panel Mounting Bracket	FNAL Drawing
61	1	534974-9	FASTBUS Module Auxiliary Connector	AMP
62	1	1-102585-3	FASTBUS Module Segment Connector	AMP
63	1		FSCC Front Panel	
64	1		FSCC P.C. Board	
65	1	DAH017	FSCC Trigger FIFO Daughter Board	
66	1	SN74AS832BN	HEX 2-Input or Drivers	TI
67	1	P82586-10MHz	Local Area Network Coprocessor	Intel
68	2	3428-5302	Male 20-Pin 100M X 100M Dip Header	3M
69	1	MC68901	Multi-Function Peripheral	Motorola
70	1	LM555CN	NE555 Precision Timer	RCA
71	4	N74F545N	Octal Bi-directional Transceiver, 3-state	Signetics
72	3	N74F1244	Octal Buffer and Driver, 3-State	Signetics
73	4	N74F646N	Octal Bus Transceiver, 3-State	Signetics
74	3	N74F825N	Octal D-Type Edge-Triggered Flip-Flops	Signetics
75	4	N74F574N	Octal D-Type Flip-Flop w/3-State Outputs	Signetics
76	1	N74F573N	Octal Transparent Latch w/3-State Outputs	Signetics
77	1	SN74LS38	Open-Collector TTL NAND Gate	Signetics
78	1	227726-1	Isolated BNC Solder Jack Assembly	AMP
80	4	LH5499-35	Parallel 4096 x 9-Bit FIFO 35 ns	Sharp
81	10	LH5496-35	Parallel 512 X 9-Bit FIFO 35 ns	Sharp
82	1	LH5496D-20	Parallel 512 X 9-Bit FIFO 20 ns	Sharp
83	1	IDT72413L45P	64 x 5-bit FIFO chip	IDT
84	2	MC68230P10	Parallel Interface/Timer	Motorola

85	1	251.010	10 Amp Subminiature Fuse	Pico
86	4	251.005	5 Amp Subminiature Fuse	Pico
87	1	251.001	1 Amp Subminiature Fuse	Pico
88	2	ITCE-5	Tranzorb for 5 and 5.2 Volt supply	
89	1		Tranzorb for 2 Volt supply	
90	1	MC10H103	Quad 2-Input or Gate	Motorola
91	1	UA96174	Quadruple Differential Line Driver	Fairchild
92	2	UA96175	Quadruple Differential Line Receiver	Fairchild
93	4		Resistor 1.5K 1/8W 5%	
94	1		Resistor 100K 1/8W 5%	
95	1		Resistor 10K 1/8W 5%	
96	1		Resistor 150 1/8W 5%	
97	17		Resistor 1K 1/8W 5%	
98	1		Resistor 1K 1/8W 1%	
99	4		Resistor 1M 1/8W 5%	
100	10		Resistor 20 1/4W 5%	
101	5		Resistor 220 1/8W 5%	
102	2		Resistor 240 1/4W 5%	
103	6		Resistor 39 1/8W 5%	
104	1	MAX233C	RS232 Driver/Receiver	Maxim
105	1	8121-S-D-A6-G-E	SPDT Push Button Switch	C&K
106	4	EPS448DC-25	Stand-Alone Microsequencer 25 MHz	Altera
107	3	SAM448-30	Stand-Alone Microsequencer 30 MHz	Waferscale
110	1	DS1386-8K	Watchdog Timekeeper, R/TClock, NVRAM	Dallas Semi.
111	8		3-pin jumper block	
112	1	DAH-017PC Rev b	Trigger FIFO Child Board Assembly	Bira Systems

6. Appendix C - FSCC Documentation

Fermilab Drawing Numbers

FNAL #	TitleDescription	
0882-MB-199070	Mounting Bracket	FASTBUS Module Front Panel Mount
0880.000-ED-215714	FSCC/VDAS Interface	E771 FSCC/VDAS interface schematic
0880.000-ED-269065	FSCC/VDAS Interface	E791 FSCC/VDAS interface schematic
0880.000-AC-269129	FSCC Layer 1	Trace Layer 1
0880.000-AC-269130	FSCC Layer 2	-2.0V & -5.2V Layer 2
0880.000-AC-269131	FSCC Layer 3	Ground Layer 3
0880.000-AC-269132	FSCC Layer 4	+5.0V Layer 4
0880.000-AC-269133	FSCC Layer 5	Trace Layer 5
0880.000-AC-269134	FSCC Top Silk	Top Silkscreen Photo
0880.000-AC-269135	FSCC Bot Silk	Bottom Silkscreen Photo
0880.000-AC-269136	FSCC Solder Mask	Top & Bot. Solder Mask
0880.000-AC-269137	FSCC Front Panel	Front Panel Silkscreen Photo
0880.000-MD-269138	FSCC Assembly	Assembly drawing
0880.000-MD-269139	FSCC Front Panel	Front Panel Mechanical Drawing
0880.000-MD-269140	FSCC Mechanical	Board Dimensions & Pads
0880.000-MD-269141	FSCC Test Interface (FSCCTI) Test Board Schematic	
0880.000-MD-269143	FSCC Schematic	11 Page Schematic and Block Diagram

Fermilab Documents

FASTBUS Smart Crate Controller - PC3, Design Specification
Fermilab Computing Division
Mark Bernett - Online and Data Acquisition Software Groups
Mark Bowden, Rick Kwarciany, John Urish - Data Acquisition Electronics Group
Fermilab Physics Department Gustavo Cancelo

Diagnostics for the FASTBUS Smart Crate Controller - PN417
Fermilab Computing Division
Mark Bernett, Dave Slimmer - Online and Data Acquisition Software Groups
Fermilab Computing Division
Richard Kwarciany, John Urish - Data Acquisition Electronics Group

Release Notes for SCG68K V2.3 - PN 376
David M. Berg, Bryan MacKinnon - Fermilab Computing Division
Online Systems Software Group

SCG68K User's Guide and Reference - PN369
Peter Heinicke, David Berg, Bryan MacKinnon, Tom Nicinski, Gene Oleynik -

Fermilab Computing Division, Online Systems and Data Acquisition Software Groups

Serial Port Driver for the PAN-DA pSOS Environment - PN379.2
Bryan MacKinnon - Fermilab Computing Division, Data Acquisition Software Group

Dart Data Acuisition System Data, Permit & Trigger Link Interface Specification - ESE-DART-950511
John Anderson, Ed Barsotti - Fermilab Computing Division, ESE Department.

Non-Fermilab Documents

IEEE Standard FASTBUS
IEEE 960
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, New York - 10017

68020 32-Bit Microprocessor Manual
Motorola Literature Distribution
P.O. Box 20912
Phoenix, AZ 85036

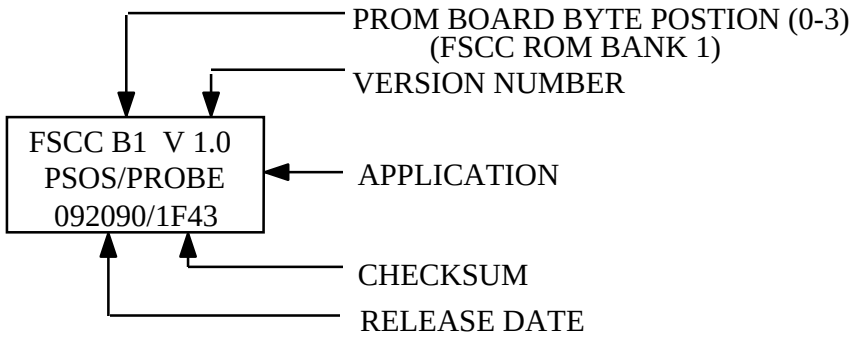
FASTBUS Standard Routines
DOE/ER 0325
National Technical Information Service, U.S. Dept. of Commerce
Springfield, Virginia 22161

FSCC Programmable Devices

FSCC Version PC4b Microcode and PLD Source Listings - HN 137
Mark Bowden, Gustavo Cancelo, Richard Kwarciany - Fermilab Computing Division, Data Acquisition Hardware Group, Online Systems Department.

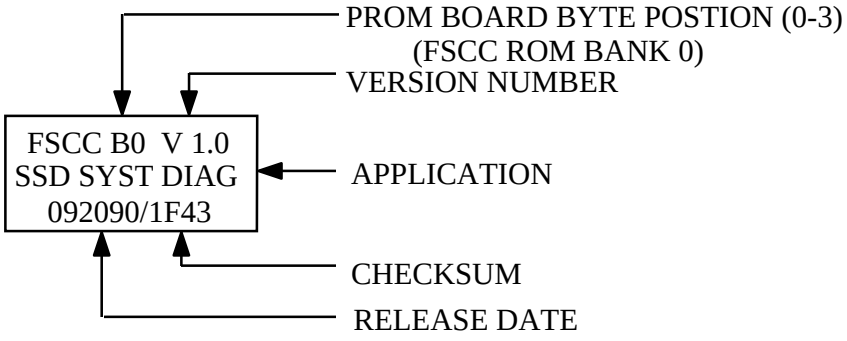
7. Appendix D - FSCC EPROM Labeling

ROM BANK 1; OPERATING SYSTEM AND FSCC DIAGNOSTICS

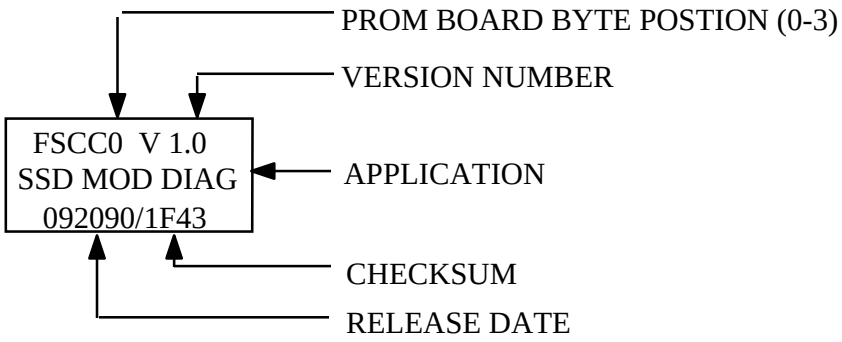


ROM BANK 2; SYSTEM OR INDIVIDUAL MODULE DIAGNOSTICS

OPTION 1; System Diagnostic Tests



OPTION 2; Individual Module Tests



8. Appendix E - FSCC PC4 Assembly Drawing

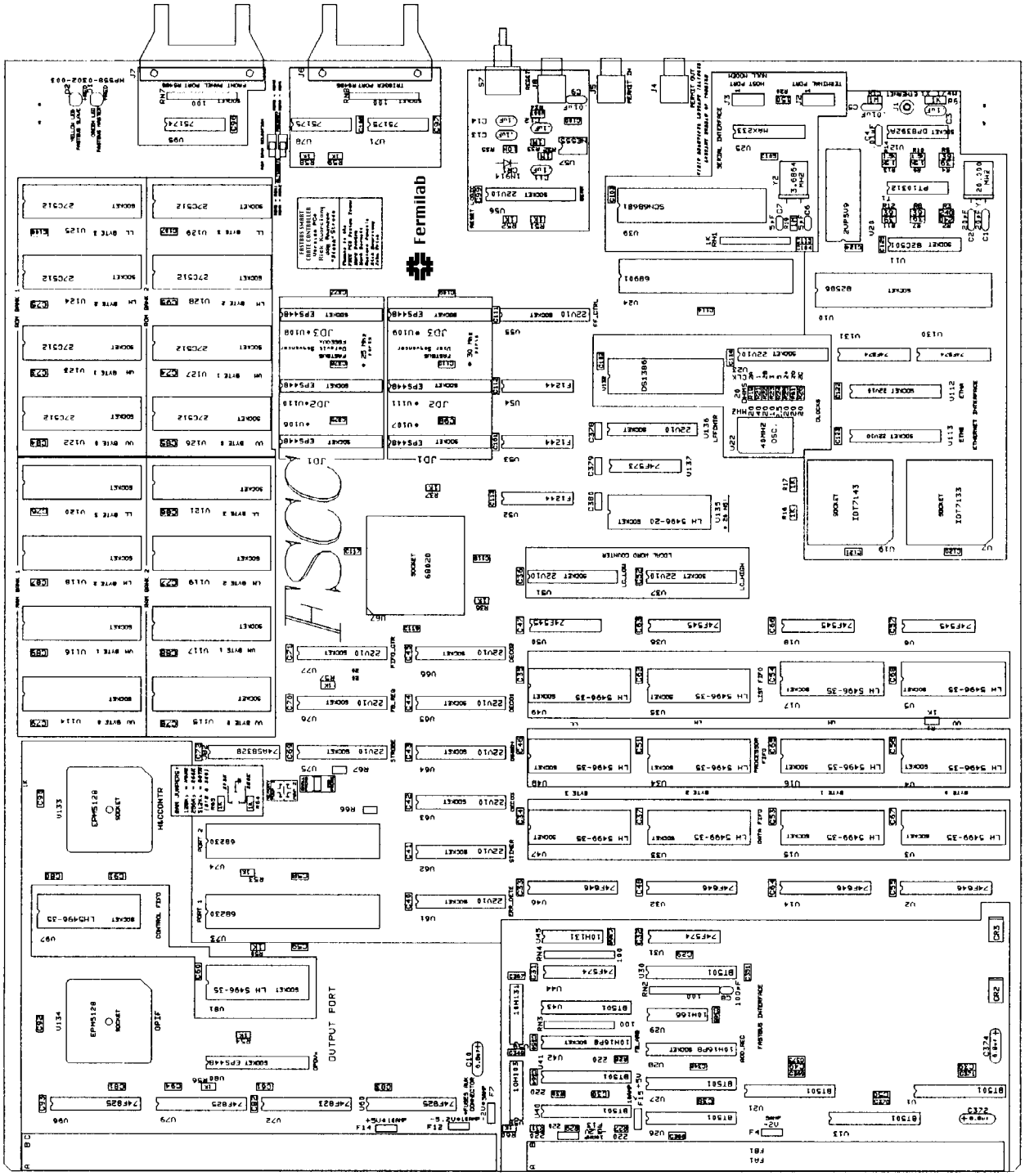


Figure 18 FSCC Component View

9. Appendix F- FSCC Version History

9.1 PC1

Prototype FSCC's were designated version PC1. A small number of PC1 FSCC's were built for testing and proof of concept purposes. PC1 modules were not widely used.

9.2 PC2 and PC3

The first production FSCC's were designated Version PC2. After some time in use, design changes were added to these modules. PC3 FSCC's were functionally identical to the modified PC2 modules. The difference is that the newer modules had the changes to the PC2 boards incorporated into the PC board layout.

9.3 PC4

9.3.1 Overview

The primary goals of the FSCC Version PC4 update project were to increase the processor's memory capacity, to replace obsolete components which were no longer in production, and to be able to complete the project in a matter of two to four months. It was also desired to maintain backward software compatibility to avoid unnecessary software redesign.

Secondary goals included expansion of Non-Volatile RAM size, increasing FASTBUS Data FIFO capacity, adding an instruction FIFO to the FASTBUS List FIFO, and bug fixes.

9.3.2 Replacement of Obsolete Components

Three of the FSCC's functions were implemented in PLD devices which have been discontinued by the manufacturer. Specifically, the EPB1400 Erasable Programmable Logic Device (EPLD) manufactured by the Altera Corporation was discontinued in 1991. The Global Word Counter & Header Latch, the Output Port Controller Interface (not shown on block diagram), and the Ethernet Port Controller to Ethernet Dual Port Memory Interface (also not shown on block diagram) are the functions affected. The Global Word counter & Header Latch, and the Output Port Controller Interface were redesigned using EPM5128 EPLD devices also manufactured by the Altera Corporation. The Ethernet Port Controller to Ethernet Dual Port Memory Interface was redesigned using conventional PALs (22V10's). All three of the redesigned circuits were completed with backward software compatibility intact.

9.3.3 Memory Expansion

9.3.3.1 Processor RAM Expansion

All FSCC's produced prior to version PC4 featured Motorola 68020 processors with 256 KBytes of zero wait state static RAM. The PC4 FSCC's are capable of having up to 4 MBytes of zero wait state RAM installed when 512K x 8-bit SRAM devices become available. Eight 256K x 8 bit devices can also be installed yielding 2 MBytes of RAM, and 128K x 8-bit devices will give 1 MByte of RAM. Switching between the different sized devices requires changing jumpers on the board, and changing PAL programming. In order to install more than 1 MByte of RAM, the memory map of the FSCC must also be changed. To make this change, jumpers on the board must be changed, and some of the processor's control PALs must be reprogrammed. Version PC4 FSCC's with the current memory map have 1 MByte of RAM installed.

The following FSCC PAL's must be reprogrammed to change RAM configurations:

DECO1 (Address Decoders)
DECO2
DECO3
DSGEN (Data Strobe Acknowledge Generator)

Table 20 FSCC RAM Configuration Options

Total RAM	FSCC Version	Memory Map	Devices Installed
128 KBytes	PC3	Version 1	4-32K x 8 bit
256 KBytes	PC3	Version 1	8-32K x 8 bit
1 MByte	PC4	Version 1	4-256K x 8 bit
2 MBytes	PC4	Version 2	8-256K x 8 bit
4 MBytes	PC4	Version 2	8-512K x 8 bit

9.3.3.2 Processor EPROM Expansion

EPROM on PC3 and older FSCC's was limited to 512 KBytes running with two wait states. PC4 FSCC's have 2 MBytes of EPROM installed with the current memory map, and are capable of having 4 MBytes total installed if the memory map was changed. EPROM on PC4 boards runs with one wait state. Changing the FSCC's memory map involves changing jumpers on the board, and reprogramming the following PALs:

DECO1 (Address Decoders)
DECO2
DECO3
DSGEN (Data Strobe Acknowledge Generator)

9.3.3.3 Processor NVRAM Expansion

Expanding the Non-Volatile RAM on the PC4 FSCC was done by replacing the DALLAS Semiconductor DS1286 NVRAM/RTC chip with a DS1386 by the same company. This device is functionally identical to the DS1286 except that it has 8 K or 32K Bytes of Non-Volatile RAM where the 1286 has only 50 bytes. The PC4 parts list currently specify the 8K devices, although the board will accept the 32K parts. Since these components are taller than a standard DIP device, it is not possible to use a socket and the NVRAM chip is permanently installed. Therefore, upgrading to the 32 K devices should only be considered when making new boards.

9.3.4 List FIFO Modifications

In order to allow the FASTBUS List FIFO to be used more easily, eight bits have been added to the List FIFO to allow FASTBUS Instructions to be stored in the List FIFO as well as FASTBUS data. This eliminates the need to write special microcode for each application where the List FIFO is used. The microcode driver needed to use the larger List FIFO is labeled "FBILSTVx" (where x is the version number), and is installed in the "User" microsequencer sockets. By default, the List FIFO and User microsequencer are disabled, and the Standard FASTBUS microsequencer is enabled. This allows the FASTBUS interface of the PC4 version of the FSCC to otherwise function identically to that of the PC3 version.

To use the List FIFO, it must be reset, loaded with a list of FASTBUS instructions, the User microsequencer must be enabled, and then the List FIFO must be enabled.

The List FIFO reset line (SRS*) is a low true signal ("0"=reset) which is connected to Parallel Port 1 bit C6. This bit must be configured as an output, then toggled low to reset the List FIFO. It must be left in a high state (PP1 bit C6 = "1") to use the List FIFO.

Example:

```
ORI.B    #$40,$0064000C *Set List FIFO Reset bit to not-reset
ANDI.B #$40,$00640004  *Configure List FIFO Reset bit as PP output
ANDI.B #$BF,$0064000C*Set List FIFO Reset bit true (reset)
ORI.B    #$40,$0064000C *Set List FIFO Reset bit false (not-reset)
```

The List FIFO Reset bit is configured as an output, and set false (not-reset) by the FSCC monitor PROBE/PSOS/DETH (V1.1 and later). Therefore, the first two instructions in the example are not necessary if this monitor is used.

To load the List FIFO, the list FASTBUS instructions are written to the FIFO in order of execution in a manner similar to normal FASTBUS instruction execution. The FASTBUS list instruction vectors (lower 10 bits of memory mapped FASTBUS instruction addresses) are identical to the instruction vectors of the standard FASTBUS instruction addresses. The address decoder portion of the FASTBUS instruction address (upper 22 bits) is the only difference.

Example:

Address	Function
00620300	Normal FASTBUS Arbitrate instruction
00600304	Normal FASTBUS Primary Address instruction
006A0300	List FASTBUS Arbitrate instruction
006A0304	List FASTBUS Primary Address instruction

A complete list of FASTBUS list instructions is included in section 2 of the FSCC Manual.

After loading, the List FIFO microcode must be initialized by enabling the User microsequencer with the following instruction:

```
MOVE.L    #$0,$006C0020        *Select User Microsequencer
```

This special microsequencer instruction is the only instruction which is executed by both the standard FASTBUS microsequencer and the user FASTBUS microsequencer. It causes the User microsequencer to enable itself, and causes the Standard FASTBUS microsequencer to disable itself. This instruction *must* be executed *after* the FASTBUS sequencer is taken out of reset, and *before* any other FASTBUS instructions are executed. Executing this instruction more than once after a reset has no effect. To return control of the FSCC's FASTBUS interface back to the standard microsequencer, the FASTBUS interface must be reset. This is done with the SNRESET* bit in Parallel Port 2, or by pressing the Reset switch on the front panel.

Once the user sequencer is selected, the List FIFO is then enabled by setting the List FIFO Enable bit high in Parallel Port 2 bit A6. Note that this bit must be configured as an output as in the following example before it can be set high.

Example:

```
**Set List FIFO enable bit to its default state, configure it as an output, then set it true**
    ANDI.B #$BF,$00660008 *Set PP2A bit 6 low
    ORI.B  #$40,$00660002  *Configure PP2A bit 6 as an output

    ORI.B  #$40,$00660008  *Set PP2A bit 6 high (List FIFO enabled)
```

The List FIFO Enable bit becoming true will allow the microsequencer to start executing the List.

The List FIFO Retransmit line is available at Parallel Port 2 bit B5. By toggling this bit list execution can be repeated without reloading the FIFO.

Example:

```
**Set List Retransmit bit to its non-active state, then configure it as a parallel port output**
    ORI.B  #$20,$00660009  *Set LRT* bit initial condition (Low true signal)
    ORI.B  #$20,$00660002  *Configure LRT* bit as an output

**Toggle List Retransmit bit true, then false to re-execute FASTBUS instruction list**
    ANDI.B #$DF,$00660009 *Set LRT* bit true
    ORI.B  #$20,$00660009  *Set LRT* bit false
```

Re-execution of the list will start with LRT* being set true.

Note that the List Retransmit bit and the List FIFO Enable bit are pulled in hardware to their default (non-active) levels. Therefore, if these bits are not configured as outputs, the FASTBUS interface will function identically to a version PC3 board. Version 1.2 and greater of the FSCC standard monitor PROBE/PSOS/DETH will have the List FIFO Retransmit and List FIFO Enable bits configured as parallel port outputs, and set to their non-active states by default. If this or a later version of the standard FSCC monitor is used, the first two processor instructions in the previous two examples may be deleted.

9.3.5 Data FIFO Modifications

PC3 FSCC's have a 2K x 32-bit FASTBUS data FIFO's installed. PC4 FSCC's have a 4K x 32-bit FASTBUS data FIFO installed. The PC4 data FIFO is otherwise functionally identical to the PC3 Data FIFO.

9.3.6 Tranzorb and Fusing Changes

In order to enhance the FSCC's ability to withstand power supply over voltage failures, PC4 FSCC's have been equipped with tranzorbs on the +5 Volt, -5.2 Volt, and -2 Volt power supplies. PC3 FSCC's also used parallel

fuses on the +5, and -5.2 Volt power supplies. These have been replaced with one larger fuse on each of these two supplies.

9.3.7 Bug Fixes

A hardware bug involving slave operation of the FASTBUS interface in PC3 and earlier FSCC's has been identified and corrected in PC4 FSCC's. The bug involves the FSCC's ability to respond to being addressed on FASTBUS. Specifically, PC3 boards have no ability to know if they have been addressed in CSR or DATA space. PC4 FSCC's have the FASTBUS line MS0 latched, and connected to Parallel Port 2 bit A5. After the FSCC has been addressed as a slave on FASTBUS, the status of PP2 bit A5 can be tested to determine if the address cycle was to CSR or DATA space. If the bit is a "1" the address cycle is to CSR space, if the bit is a "0" the address cycle is to DATA space.

Two microcode bugs have also been identified in the standard FASTBUS microcode FBSEQV2 dated 9-24-90. The first problem is that FASTBUS GK is not cleared by either a FASTBUS sequencer reset, or a hard reset. This bug has been fixed in FBSEQV2 dated 8-18-92 which is installed on all PC4 FSCC's. The second bug involves an incorrect MS code during Broadcast CSR Space primary address cycles. This bug has also been corrected in the current version of the microcode. This version of the standard microcode can also be installed on PC3 FSCC's.

9.4 PC4a

9.4.1 Overview

The primary goals of the PC4a update, were to modify the FSCC to enable it to work more effectively in the DART DAQ architecture, to add memory and other features to enhance programming ease, and to correct hardware bugs which were discovered after the PC4 was produced.

9.4.2 Front Panel Trigger Port Enhancements

In order to take advantage of the ability of some front end modules to internally buffer events, the FSCC needed to be able to buffer event triggers, and trigger ID's. To this end, a 64 word deep Trigger FIFO was added to the Front Panel Trigger Input Port. Trigger strobes received at the port now connect to the Trigger FIFO's "Shift In" input. Trigger ID values at the port are then clocked into the Trigger FIFO. The FPORT controller now monitors the Trigger FIFO's "Output Ready" output, instead of the Trigger Strobe input directly. The Trigger_Strobe_Hold FPORT instruction tests the Output Ready line, and waits for it to go true. When this happens, it proceeds with its readout instruction list. Presumably, this list would contain an End_Of_Event instruction which will cause the OPORT controller to start outputting the event. The End_Of_Event instruction also clocks the next Trigger ID out of the Trigger FIFO, so the proper Trigger ID will be attached to each event. The data outputs of the Trigger FIFO containing the Trigger ID values, now connect directly to the Header & Counter logic, allowing the Trigger ID to be automatically written directly into the Header field of the leading word count word. This feature of automatically writing the Trigger ID values directly into the Header may be enabled by setting a bit in the H&C Control register.

In order to prevent the Trigger FIFO from being overrun by triggers, a Trigger Hold Off output is provided. This output is jumper configurable to go true on one of four possible conditions. Trigger FIFO Almost Full (56 or more Triggers are queued), Trigger FIFO Half Full (32 or more Triggers are queued), Trigger FIFO Empty, and End_Of_Event (signals that the event readout is complete).

9.4.3 Suppressing Zero Word Events (SZE)

In some systems, it is possible for the front end modules to not have data after some events. If the FSCC's FPORT controller is running in a loop where it reads out the crate and outputs the data after each trigger is received, the OPORT controller would output a Word Count Word with a value of zero, and then no data. In order to reduce the amount of meaningless data collected, it is sometimes desirable to suppress these "Word Count only" events. The ability to suppress these events is provided for in PC4a FSCC's. When enabled, if the OPORT controller pulls a zero word event from the Data FIFO, it will simply pass the token without outputting the zero word event. Enabling the SZE feature does not effect OPORT output rate or operating mode settings. The feature is enabled by setting a control bit in the OPORT Control register. The Reset default state of the SZE bit is zero (disabled).

9.4.4 Write Protect Non-Volatile RAM

To protect the Non-Volatile RAM and the Real Time Clock (RTC) from inadvertently being overwritten, a write protect/enable bit has been added. By default, the NVRAM/RTC is Read-Only. To write to the NVRAM/RTC, the write enable bit must be set. The NVRAM and the Real Time Clock can then be accessed normally until the write protect bit is set.

9.4.5 FPORT Microcode Enhancements

Six new microcode instructions have been added to PC4a modules. Five of these instructions are List Mode instructions, and one is a Normal FPORT instruction. Three of the five List Mode instructions are delay instructions. The FPORT can now be programmed to pause for a given amount of time before executing the next instruction. This is done by inserting a delay instruction at the desired point, into the list like any other instruction. A pause for two, ten, or one hundred microseconds is generated upon execution of one of the three instructions. Delay instructions can also be chained in any order.

A POLL_HALT_REQUEST (PHR) instruction has also been added. This List FPORT instruction is inserted into the list at a convenient stopping point, to allow a graceful halting of FPORT List execution when the List is repeating itself. Each time the FPORT executes the PHR instruction, it checks to see if an FPORT List Halt Request instruction has been executed by the processor. If the processor hasn't executed such an instruction, then list execution continues. If the processor has requested a List Halt, then the FPORT halts list execution. The processor FPORT List Halt Request instruction can only work if the PHR instruction is executed by the FPORT. If the processor Halt Request instruction is executed, but the FPORT does not execute a PHR instruction within the Long Time-out period, a 68020 bus error exception will occur. The FPORT must be reset to take it out of the halted state.

The GENERATE_FPCREQ instruction generates an FPCREQ interrupt when executed. If it is desired to cause a CPU interrupt at some point during list execution, this list instruction is inserted at the desired point in the list. The FPCREQ interrupt is then enabled in the interrupt controller. Some care must be exercised in the use of this instruction, since if the GENERATE_FPCREQ instruction is in a list which is repeating itself, it is possible to overwhelm the operating system with interrupts, causing an apparent CPU hang. In general, use of this instruction in a self repeating list should be avoided, unless it is known that the minimum time between interrupts will be long enough for the CPU to service them.

In order to facilitate the use of FSCC's in DAQ system diagnostics, it was desired to be able to transmit the same data out of the FSCC's Data FIFO many times without reading out any slaves. This is possible by first filling the Data FIFO, and executing an End_Of_Event instruction to output the first data block. Then toggling the Data FIFO Retransmit bit, and executing the End_Of_Event instruction, instead of filling the Data FIFO each subsequent time the data is to be output. This bit sets the Data FIFO's internal pointers to zero, so that the previously outputted data can be output again. A problem arises when an End_Of_Event instruction is executed to output the event. The End_Of_Event instruction tells the OPORT controller to start outputting data, but it also writes the End_Of_Event flag into the back of the Data FIFO to delineate events in the FIFO. The OPORT controller then pulls the data out of the FIFO and outputs it until it sees the End_Of_Event flag pop out. Under normal use, when the Data FIFO is being filled by reading out front end boards over FASTBUS, this is no problem, but if the Data FIFO is being Retransmitted, then there will already be an End_Of_Event flag in the FIFO from the previous event. Now when the End_Of_Event instruction is executed to tell the OPORT to start outputting the data, another End_Of_Event flag is inserted into the Data FIFO. If the test program is looping where it Retransmits the Data FIFO and then executes an End_Of_Event instruction, the Data FIFO will eventually fill up with End_Of_Event flags, causing an FPORT error. To prevent this from happening, a special version of the End_Of_Event instruction was included on PC4a boards. This instruction functions identically to the normal End_Of_Event instruction, except that it does not write an End_Of_Event flag into the Data FIFO. This Normal (Non-List) FPORT instruction is called End_Of_Event_Rexmit.

9.4.6 Add Control FIFO Status Bit

To give the CPU, and the user more information about what the FSCC is doing at the current time, the Control FIFO Empty flag has been mapped into an OPORT status register. When this bit reads as a zero, this means that there are no Header/Word Count words in the Control FIFO. When an event is read into the FSCC's Data FIFO, and an End_Of_Event instruction is executed, the Header/Word Count word is inserted into the Control FIFO, causing the Status bit to read high. As soon as the OPORT controller starts outputting the data, it pulls the word out of the Control FIFO, and the Status bit will go low again. Effectively, the Status bit being true means that there is at least one complete event in the Data FIFO which is not yet being output. If the Status bit reads low, this does not mean that there is no Data in the Data FIFO, it may contain a part of an event which is currently either being read out, or being output.

9.4.7 Modify OPORT Controller to Comply with DART Protocol

The DART data stream protocol requires that data blocks on the data stream cable be delineated by a control signal pulse. This control signal is called End_Of_Record (EOR). The PC4a OPORT controller can drive EOR after it outputs the last data word, and before it passes the token, if it is operating in Event_EOR mode.

The PC4a OPORT controller has also been enhanced by adding a variable data output rate feature. By default, the OPORT outputs data at a 10 MHz rate (40 MBytes / Sec). If desired, this rate can be reduced through software. The two other supported data rates are 6.67 MHz (26.68 MBytes / Sec), and 5.0 MHz (20 MBytes / Sec).

9.4.8 Expand CPU Memory Map

In order to allow the use of the second band of PROM, and to allow the installation of larger RAM chips to expand available program RAM, the CPU memory map was changed to allow for larger RAM and ROM areas.

9.5 PC4b

9.5.1 Overview

The primary goal of the PC4b modifications were to change the FSCC to allow it to meet the newly revised DART (Data Acquisition Real Time) Interface Specification (Document Number: ESE-DART-950511-A). The DART Interface Specification defines the physical medium, data format, and timing of the RS-485 data stream, PERMIT_IN/PERMIT_OUT token passing links, and the Event Trigger Link.

Secondary goals of the PC4b changes were to enhance data flow control by routing the Data FIFO status flags into the Trigger Hold-Off Output logic, and to fix two design flaws.

9.5.2 DART Interface Specification Changes

9.5.2.1 Data Link Changes (OPORT)

Most of the changes required to meet the DART Data Link Specification, actually involved changes to the auxiliary card (FSCC-DARTAC). However, there were some changes to the FSCC itself. The timing of the enable lines to the RS-485 drivers on the FSCC-DARTAC was changed to enable the driver for the Data Strobe line 50ns before the drivers for the data lines. This was done to help prevent false Data Strobe transitions due to crosstalk on the data cable. The OPORT controller was also changed to prevent data transitions for 100ns after the data drivers are enabled.

Supported OPORT modes have been redefined to more realistically reflect the way the OPORT is actually used. The designation ***Event_Mode*** has been renamed ***Token_Middle***. ***Force_Event_Mode*** is now known as ***Token_First***, and ***Event_EOR*** is now ***Token_Last***. The name changes are to more accurately reflect the function of the modes, and hopefully, to make programming the FSCC somewhat more intuitive.

Token_Only mode has been added to allow the use of the PC4b as the only data source on a data link. With previous versions of the FSCC, it was sufficient to set the OPORT into Event_Mode (now Token_Middle), and use a LEMO type terminator installed in the PERMIT_IN input. This effectively forced the PERMIT_IN input true. With the conversion of the PERMIT_IN input to NIM level, putting a terminator into the PERMIT_IN input does not force the input true, therefore, the new mode was needed.

CPU mode (Control mode) operation of the OPORT is no longer supported. CPU mode allowed data to be written directly into the Control FIFO, and output as data through the OPORT. This mode was originally thought to be useful for testing the personality cards, and the data links connected to them. However, it has become obvious that writing data directly into the Data FIFO is much more useful for testing purposes. To make space in the OPORT controller for Token_Only mode, the seldom used CPU mode was deleted.

Event_With_Manual_Permit_Out and Permit_Out modes are also no longer supported. These modes were originally designed to allow the readout program to drive End_Of_Record and other control lines manually from the processor. The PC4b OPORT has been enhanced so that End_Of_Record is now driven with the appropriate timing when the OPORT is set into Token_Last mode. Once configured, the readout code no longer needs to be concerned about the communication protocol on the data link.

The RS-485 driver for the End_Of_Record signal is now enabled whenever the OPORT is set into Token_Last or Token_Only modes. The driver is disabled in all other modes.

The FSCC-DARTAC was changed to allow the Data Strobe pulse to be inverted and lengthened to match the DART Data Link Specification. The active edge is now the low going edge of the Data Strobe. All termination SIP resistor packs were removed from the FSCC-DARTAC to allow the use of DART Upstream and Downstream termination modules, and unused control lines 16/32*, SSTROBE, FEVEN, SPARE0, SPARE1, and SPARE2 were either cut or permanently disabled.

The function of the four DIP switches on the FSCC-DARTAC have been redefined. Switch 1, 3, and 4, now have no function, and can be left in either the open or closed position. Switch 2 now controls the RS-485 WAIT enable.

When the switch is open, RS-485 WAIT is disabled. WAIT received from the data buffer is ignored. When the switch is closed, WAIT is enabled. WAIT received from the data buffer is passed to the FSCC and the data transmission is paused. This switch setting was added to allow the RS-485 cables to be disconnected and still allow data to be clocked out of the OPORT for testing purposes.

9.5.2.2 PERMIT Link Changes (PERMIN/PERMOUT)

The most significant change to the front panel PERMIT connectors is that they were changed to NIM level signals from TTL level signals. The active level for both PERMIT_IN, and PERMIT_OUT, is a NIM "1", which approximately -0.8 Volts when the line is terminated in 50 ohms. The inactive level is NIM "0" which is approximately ground level. The FSCC's PERMIT_IN connector supplies the necessary 50 ohm NIM termination. PERMIT_IN and PERMIT_OUT were also redefined to be edge triggered signals rather than level triggered.

In order to use the FSCC as the only data source a data cable, the module must think that it is both "first" and "last" on the PERMIT token passing chain. It must be first so that it starts outputting data without waiting for a PERMIT_IN on the first event, and it must be last so that it drives the EOR signal on the data cable. With previous versions of the FSCC, "first and last" mode was done by setting the FSCC into Token_Last (formerly Event_EOR) mode, then putting a 50 ohm terminator into the PERMIT_IN input on the front panel. The terminator forced the PERMIT_IN input to its active state, so the module always had the token. Since the PERMIT connections were changed to NIM levels on the PC4b FSCC's, inserting a 50 ohm terminator into the PERMIT_IN input no longer forces the input to its active state. To allow the PC4b modules to be both "first and last" in the PERMIT chain, a new mode was added to the Output Port Controller. When using the PC4b FSCC as the only data source on the data cable, ***Token_Only*** mode should be used.

9.5.2.3 Trigger Link Changes (Trigger Strobe, and Trigger ID bits)

The RS-485 Trigger Strobe input on the PC4b FSCC has been changed slightly. The active edge of the RS-485 Trigger Strobe has been changed to the low-going edge. The termination resistors have been removed from both the RS-485 Trigger Strobe and the Trigger ID inputs. An external termination module must now be used. A NIM level version of the Trigger Strobe input was also added to the PC4b FSCC which is logically ORed with the RS-485 Trigger Strobe input. This input is also on the front panel.

The Trigger Hold Off (THO) front panel output has been changed from a TTL level signal to a NIM level signal. The active level for all front panel NIM level signals is a NIM "1" (approximately -0.8 Volts when terminated by a 50 ohm resistor).

9.5.3 Data Flow Control Enhancements

Originally, the FSCC was not designed with multi-event buffering front end modules in mind, for the simple reason that they did not exist at that time. Since it was desired to use the FSCC with these newer front-end modules, the Trigger FIFO and the Trigger Hold Off (THO) output was added to the FSCC during the PC4a upgrade. In an effort to prevent data buffer overflow when the FSCC is used with these modules, two of the FSCC's Data FIFO status flags have been added to the THO logic. Data FIFO Half Full, and a latched version of Data FIFO Full have been logically ORed with the jumper selectable THO condition available on the Trigger FIFO child board. If the Data FIFO becomes half full during a readout, the THO output is driven true until the Data FIFO becomes less than half full. If the Data FIFO becomes full during a readout, this error condition is latched by the THO logic and the THO output is driven true until the Output Port Controller is reset.

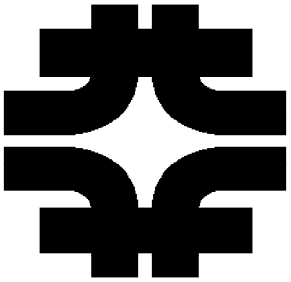
9.5.4 Bug Fixes

Two bugs have been corrected in the PC4b modifications. The first involved skewing of the FASTBUS Data Acknowledge (DK) signal. The FASTBUS specification dictates that the Master must delay the incoming DK signal from the Slave by a small amount, to allow some setup time for the Slave Status (SS) lines. Previous versions of FSCC's did not have this delay, and it was noticed that there were rare but persistent problems with the FSCC when used with certain slaves. The DK input of PC4b FSCC's now provide this delay.

The second bug is related to the FASTBUS Short Timer time-out value. The Short Timer is the device which times the response of the Slave or Ancillary Logic to the Master's assertion of certain control lines. Previous version of the FSCC had a Short Timer time-out value of approximately 1.6 micro-seconds which is the minimum allowed by the FASTBUS specification. It seems that some Ancillary Logic had Broadcast timers of approximately the same value. This caused occasional time-out errors during Broadcast addressing cycles. The time-out value of the Short Timer on the FSCC was increased to prevent this error.

During testing of certain front-end modules which featured Mega-Block mode readout, it was noticed that the readout worked without errors until a certain number of modules were added to the Mega-Block chain. This caused a Short Time-out error on the FSCC. This problem was traced to the way the Slaves released the AS-AK lock after the Mega-Block readout was completed. The release of AK had to ripple back through all of the Slaves before the AK line on the bus was lowered. Each of the Slaves added some delay, until the total AS(down) to AK(down) time was greater than the newly lengthened Short Timer value on the FSCC. Since the current FASTBUS specification is somewhat vague with regard to this condition, it was decided that lengthening the FSCC's Short Timer value even more was the most cost effective solution. The PC4b Short Timer value is approximately 3 micro-seconds.

10. Appendix G - FSCC Auxiliary Output Port Interface Cards



Fermi National Accelerator Laboratory

10.1 FSCC- DARTAC INTERFACE

AUXILIARY BOARD

Version 3.1

Oscar Trevizo, Mark Bowden, Jeff Constable,
Geoff Cottrell, Rick Kwarciany, Dan Moline

October 4, 1995

10.1.1 GENERAL INFORMATION

The *FSCC-DARTAC Auxiliary Board* is a custom version of the original *FSCC-VDAS Auxiliary Board* . It will interface the FASTBUS Smart Crate Controller (FSCC) with the FIFO system of DART Data Acquisition System compliant buffers, to allow data flow *from* the FSCC *to* the attached DAQ system. Connector requirements for the DART system are provided for in this version.

Figure 19 shows how data flows from the FSCC to a DAQ buffer through the auxiliary board:

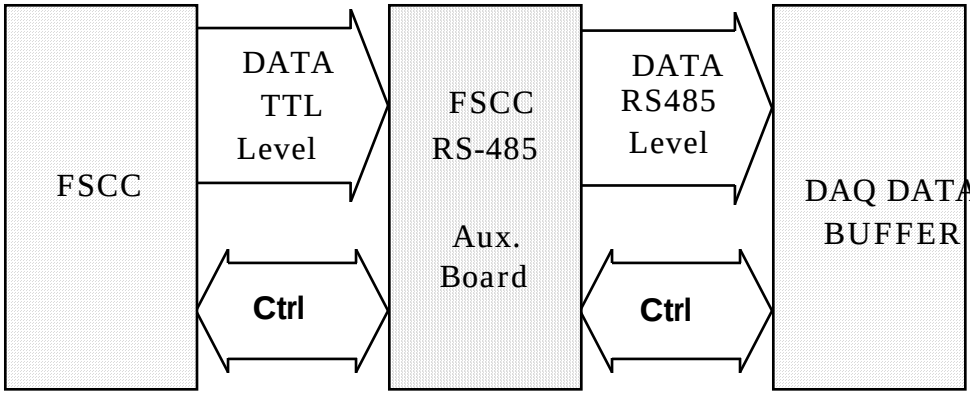


Figure 19 DART DAQ System Block Diagram (Partial)

10.1.1.1 Board Purpose

The *FSCC-DARTAC Auxiliary Board* converts TTL level signals coming from the FSCC output port (FASTBUS Auxiliary connector) to RS485 level signals. The FSCC output port is a 195 pin FASTBUS standard 3 row connector containing 32 bits of data, and 6 control signals. Pin definitions of the connector are shown in this document.

Two twisted pair ribbon cables connect data and control signals on the auxiliary board to the DAQ system. A 50 pin ribbon cable for the lower 16 bits of differential data, control signals, and a 34 pin cable for the upper 16 bits of differential data.

10.1.1.2 Packaging

This is a standard FASTBUS auxiliary board. FASTBUS auxiliary boards are located in the auxiliary backplane port of the crate. See *IEEE STD 960-1986* section 14.

10.1.1.2.1 Physical Size

Physical dimensions of the board comply with FASTBUS auxiliary boards. See section 14 of *IEEE STD 960-1986.*

10.1.1.3 Power Requirements

Power is supplied by connection to the FSCC through the FSCC's FASTBUS Auxiliary connector. The board requires +5 Volts @ 1.3 Amp.

10.1.1.4 Cooling Requirements

Cooling requirements comply with FASTBUS standards (see *FASTBUS Std* section 13.3).

10.1.1.5 ICs Used

1. National Semiconductor chips DS96174 and DS96175, 16 pin Quad Differential Line Drivers, make the level conversions between TTL and RS485. They meet a transmission rate of 10 Mbs. The following is a list of specs:

- •Meets EIA Standard for RS485 and RS422A
- •Monotonic Differential Output Switching
- •Three-State Outputs
- •Designed For Multiple Bus Transmission
- •Common Mode Output Voltage Range: -7V to +12V
- •Operates From Single +5V Supply
- •Thermal Shutdown Protection

For more information see National Semiconductor Linear Data Book.

2. TTL 74F00 converts proper polarity for strobe signal. This gate also allows the delay path of the Strobe line to be similar to the delay path for the data.

3. Lattice GAL22V10-15 PAL currently used for miscellaneous combinatorial logic.

10.1.1.6 Pin Configurations

The three connectors used are: a 195 pin 3 row connector from the FASTBUS auxiliary backplane, a 50 pin 3M connector and a 34 pin 3M connector for DART buffers.

10.1.1.6.1 FASTBUS 195 Pin 3 row Backplane Connector

For standard auxiliary backplane connector see FSCC documentation.

Pins B16 through B47 hold data(0) through data(31) with B59 for DATA_OUTPUT_ENABLE.

Control pins are B14 for STROBE, B58 for WAIT, and B15 for End_Of_Record.

Driver enables are as follows: B59 for Data_Output_Enable, B60 for Strobe_Output_Enable, and B8 for End_Of_Record_Enable.

Power pins are: A12, A32, and C63 for -5.2V; A43, C12, and C53 for +5V; B65 for -2V; and A22, A53, A63, B64, C22, C32, and C43 for GROUND.

10.1.1.6.2 50 Pin Connector

This connector contains data as follows:

Pin1 has data(0), Pin2 has -data(0)
Pin3 has data(1), Pin4 has -data(1)
Pin5 has data(2), Pin6 has -data(2)
Pin7 has data(3), Pin8 has -data(3)

Pin9 has data(4), Pin10 has -data(4)
Pin11 has data(5), Pin12 has -data(5)
Pin13 has data(6), Pin14 has -data(6)
Pin15 has data(7), Pin16 has -data(7)
Pin17 has data(8), Pin18 has -data(8)
Pin19 has data(9), Pin20 has -data(9)
Pin21 has data(10), Pin22 has -data(10)
Pin23 has data(11), Pin24 has -data(11)
Pin25 has data(12), Pin26 has -data(12)
Pin27 has data(13), Pin28 has -data(13)
Pin29 has data(14), Pin30 has -data(14)
Pin31 has data(15), Pin32 has -data(15)
Pin33 has strobe, Pin34 has -strobe
Pin35 is not used, Pin36 is not used
Pin37 has WAIT, Pin38 has -WAIT
Pin39 is not used, Pin40 is not used
Pin41 has EOR, Pin42 has -EOR
Pin43 is not used, Pin44 is not used
Pin45 is not used, Pin46 is not used
Pin47 is not used, Pin48 is not used
Pin49 is not used, Pin50 is not used

10.1.1.6.3 VDAS 34 Pin Connector

Pin1 has data(16), Pin2 has -data(16)
Pin3 has data(17), Pin4 has -data(17)
Pin5 has data(18), Pin6 has -data(18)
....
....
Pin31 has data(31), Pin32 has -data(31)

10.1.2 THEORY OF OPERATION AND OPERATING MODES

Each of eight DS96174 converts four bits of TTL level data to differential RS485 level data. One DS96174 converts the STROBE signal, to RS485 level. The DS96175 converts the WAIT RS485 level signal from the DART buffer module to the TTL WAIT signal on the FSCC. The Strobe output of the FSCC is run through a gate to allow its delay path to more closely match the delay path of the Data. Data is valid on the leading (falling) edge of the RS-485 Strobe signal The 22V10 PAL on the board was added to version 3.0 to allow more implementation flexibility concerning the WAIT input, and also the EOR (a.k.a.'s: EOE or EOB) output. The pal also has a four switch DIP switch connected to it to allow quick configuration changes in the field. For detailed operation of the WAIT input logic, and of the EOR output logic see PAL equation listing at end of this document. Current DIP switch definitions are listed in their own section of this document.

10.1.2.1 Basic Operation

The basic operation is better described by the timing diagram in section *2.11*, however, a summary is provided:

- A 100ns clock drives a the 32-bit data register on the FSCC.
- STROBE from the FSCC rises ~40 nsecs. after data becomes valid.
- STROBE is inverted and delayed slightly. Data is valid on the falling edge.
- New data is presented on 100 nsecs cycles.

This process continues until the DAQ buffer asserts WAIT, or the entire event is output. During WAIT:

- The data register clock to stays LOW.
- STROBE to stay LOW (RS-485 data strobe remains high).
- Data does not change.

10.1.2.1.1 DIP Switch Settings

Function	Switch 1
No Function	X

Function	Switch 2	
FSCC WAIT Always False	open	
FSCC WAIT = RS-485 WAIT	closed	*

Function	Switch 4	Switch 3
No Function	X	X

** = Default setting for DART System.*

DIP Switch 1 has no function, and may be left in either position.

DIP Switch 2 allows the RS-485 WAIT control line to be either received or ignored by the FSCC.

DIP Switches 3 and 4 have no function and may be left in either position.

10.1.2.1.2 Jumper Settings

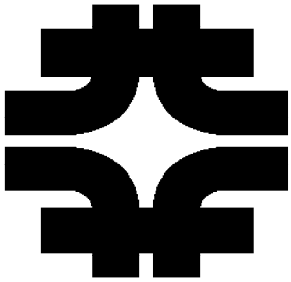
The jumper block on the FSCC-DARTAC Aux. card has no function. The jumper may be in any position or removed.

Table 21 FSCC-DARTAC Parts List

Description	Quan	Stock #	Manufacturer	Man. #	Price each	Cost/board
.1uF Ceramic, Dip Cap	14	1415-3140	Sprague	923C25U104M050B	$0.20	$2.80
100 Ohm, 8-pin, 4-resistor SIP	11		Bourns	4308-102-101	$0.20	$2.20
47uF 25V Polarized Capacitor	1	1425-1200			$0.98	$0.98
50ns Delay	1		Dallas	DS1000-50	$3.98	$3.98
8-Pin SIP Socket	11		Samtec	SS-108-G2	$0.85	$9.35
Circuit Board	1		(by contract)		$45.00	$45.00
FASTBUS AUX connector	1		AMP	534974-9	$27.00	$27.00
Miniature fuse 2 A	1		PICO	251.002	$0.75	$0.75
Right angle header, 34-pin	1	1435-7105	3M	3431-5302	$1.60	$1.60
Right angle header, 50-pin	1	1435-7115	3M	3433-5302	$2.25	$2.25
Quad 2-input Positive Nand	1		TI	N74F00N	$0.40	$0.40
Quad Differential Line Driver	10		National	DS96174	$1.75	$17.50
Quad Differential Line Receiver	1		National	DS96175	$1.60	$1.60
Transorb	1			ICTE-5	$1.24	$1.24
22 input, 10 output PAL	1		Lattice	GAL22V10-15	$14.00	$14.00
4 switch DIP switch	1					$0.00
1K Ohm 1/8 Watt Resistor	4					$0.00
2x3 pin jumper block	1					$0.00
Miniature Spring Socket	2		AMP	2-331272-2	$0.23	$0.46
24-pin x 300Mil DIP socket	1		Samtec	ICO-324-SGG	$1.87	$1.87
					Total	$132.98

10.1.2.1.3 PAL Source Listing

```
module auxcard
title 'DART auxcard WAIT, and EOR logic
     Richard Kwarciany  Fermilab 4/28/93'

" Made changes to meet DART Interface 3.11 spec.
"       Add EOR_OE equation
"       Eliminate DIP switch setting for EOR source
"       RK  6-14-95

     auxcard     device      'p22v10';

CLK              PIN   1; "AC07
!SWAIT           PIN   2; "wait input from RS-485
!OPOEOR          PIN   3; "EOR from FSCC OPORT controller (AC09)
OUTEN            PIN   4; "RS-485_Output Enable
STROBE           PIN   5; "strobe from FSCC
SPARE0           PIN   6; "Spare input from RS-485 (Formerly Sequencer Busy)
SPARE1           PIN   7; "Spare input from RS-485
AC12             PIN   8; "spare input from FSCC OPORT sequencer
DIP1             PIN   9; "DIP switch EOR source sel
DIP2             PIN   10;"DIP switch WAIT sel
DIP3             PIN   11;"DIP switch EOR enable sel 0
DIP4             PIN   13;"DIP switch EOR enable sel 1
AC02             PIN   14;"spare I/O from FSCC processor
AC06             PIN   15;"spare I/O from FSCC processor
AC05             PIN   16;"spare I/O from FSCC processor
EOR_OE           PIN   17;"EOR output enable
!PROEOR          PIN   18;"EOR from FSCC processor
!DEOR            PIN   19;"EOR output to RS-485
SPARE2_OE        PIN   20;"output enable for SPARE2 RS-485 output
DEOR_OE          PIN   21;"output enable for EOR output to RS-485
SPARE2           PIN   22;"spare RS-485 output
!FWAIT           PIN   23;"wait output to FSCC OPORT controller



equations

" Drive the EOR output using the input from the FSCC's OPORT controller.
DEOR       =     OPOEOR;

" Drive FSCC WAIT if BUFFER_BUSY is recieved
"    or tie wait perminantly false by setting DIP2 to a one (switch open).
FWAIT      =     !DIP2 & SWAIT;

" EOR output is enabled by the FSCC's output port controller.
DEOR_OE    =     EOR_OE;

" SPARE2 output is perminantly disabled.
SPARE2_OE  =     0;

" SPARE2 is unused.
SPARE2     =     0;

END
```



Fermi National Accelerator Laboratory

10.2 FSCC-VDASAC INTERFACE (E791)

AUXILIARY BOARD

Version 3.0

Oscar A. Trevizo, Mark Bowden
Jeff Constable, Richard Kwarciany, Dan Moline, Geoff Cottrell

October 31, 1992

10.2.1 1.GENERAL INFORMATION

The *FSCC-VDAS Auxiliary Board* will interface the FASTBUS Smart Crate Controller (FSCC) with the FIFO system of the Video Data Acquisition System (VDAS) to allow data flow *from* the FSCC *to* the VDAS-FIFO .

FSCC is a simple readout controller for low occupancy front-end modules that performs most basic FASTBUS operations. VDAS, is a memory handling FIFO system with a 32-bit input port based on the RS485 bus standard. It provides the mechanisms to store amounts of data as large as 176 MBytes at speeds of 40 MBytes/sec in and 40 MBytes/sec out, simultaneously.

Figure 20 shows how data flows from the FSCC to the VDAS-FIFO through the auxiliary board:

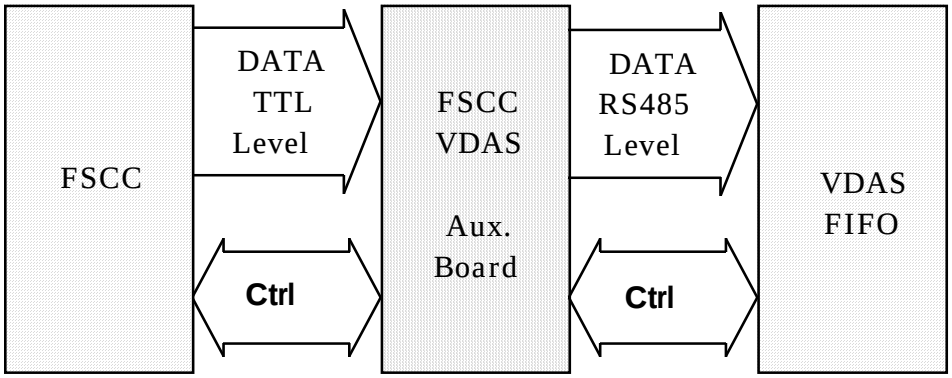


Figure 20 VDAS DAQ System Block Diagram (Partial)

Fermilab experiment E-791 is scheduled to use the FSCC-VDAS combination to collect data from fixed-target experiments.

10.2.1.1 Board Purpose

The *FSCC-VDAS Auxiliary Board* converts TTL level signals coming from the FSCC output port to RS485 level signals going into the VDAS input port. The FSCC output port is a 195 pin FASTBUS standard 3 row connector containing 32 bits of data, STROBE, and WAIT control signals. The pin definitions of the connector are shown in this document.

VDAS input port contains two connectors; a 64 pin ribbon connector for 32 bits of true and inverse data, and a 10 pin connector for STROBE and WAIT control signals.

The auxiliary board will pass data to the VDAS system as long as no WAIT signal is generated by the VDAS system. The VDAS system will send a WAIT signal to the auxiliary board in the event that a FIFO "near full" condition occurs.

10.2.1.2 Application

The first application for this board is for Fermilab experiment E-791, *Hadroproduction of Charm and Beauty* in the Tagged Photon Laboratory. The main objective of E-791 is to explore new ground in charm and beauty physics. Fast front-end electronics (<20 μs readout times) and fast data acquisition is required

as part of the overall system for reconstructing a large number of events in E-791. For more information about E-791 see the *1989 Fermilab Research Program Workbook* page 95.

10.2.1.3 Packaging

This is a standard FASTBUS auxiliary board. FASTBUS auxiliary boards are located in the auxiliary backplane port of the crate. See *IEEE STD 960-1986* section 14.

10.2.1.3.1 Physical Size

Physical dimensions of the board comply with FASTBUS auxiliary boards. See section 14 of *IEEE STD 960-1986.* (Roughly 5" wide by 7" tall, and .093" thick)

10.2.1.4 Power Requirements

The card is powered by a connection to the FSCC through the FSCC's Auxiliary connector. Total power consumption is approximately 2.5 Watts on the +5Volt supply.

10.2.1.5 Cooling Requirements

Cooling requirements comply with FASTBUS standards (see *FASTBUS Std* section 13.3).

10.2.1.6 Integrated Circuits Used

Fairchild chips µA96174 and µA96175, 16 pin Quad Differential Line Drivers, make the level conversions between TTL and RS485. They meet a transmission rate of 10 Mbs. The following is a list of specs:

- Meets EIA Standard for RS485 and RS422A
- Monotonic Differential Output Switching
- Three-State Outputs
- Designed For Multiple Bus Transmission
- Common Mode Output Voltage Range: -7V to +12V
- Operates From Single +5V Supply
- Thermal Shutdown Protection

For more information see Fairchild Linear Data Book 1987 page 9-87.

2. Delay IC from Dallas Semiconductor DS 1000-50 used to make a "one-shot" for the strobe signal.

3: TTL 74F00 converts proper polarity for strobe signal.

10.2.1.7 Pin Configurations

The three connectors used are: a 195 pin 3 row connector from the FASTBUS auxiliary backplane, a 64 pin connector and a 10 pin connector for VDAS.

10.2.1.7.1 FASTBUS 195 Pin 3 row Backplane Connector

For standard auxiliary backplane connector see FSCC documentation section 1.32.

Pins B16 through B42 hold data(0) through data(31) with B59 for OUTPUT-ENABLE.

Control pins are B14 for STROBE*, and B56 for WAIT.

Power pins are: A12, A32, and C63 for -5.2V; A43, C12, and C53 for +5V; B65 for -2V; and A20, A53, A63, B64, C22, C32, and C44 for GROUND.

10.2.1.7.2 VDAS 64 Pin Connector

This connector contains data as follows:

Pin1 has -data(31), Pin2 has +data(31)
Pin3 has -data(30), Pin4 has +data(30)
Pin5 has -data(29), Pin6 has +data(29)
....
....
Pin61 has -data(1), Pin62 has +data(1)
Pin63 has -data(0), Pin64 has +data(0)

10.2.1.7.3 VDAS 10 Pin Connector

Pin10 has +STROBE
Pin9 has -STROBE

Pin6 has WAIT ("near full")
Pin5 has -WAIT.

10.2.2 THEORY OF OPERATION AND OPERATING MODES

The board contains nine µA96174, one µ96175, one 195 FASTBUS backplane connector, one 64 pin ribbon connector, and one 10 pin connector. Each of eight µA96174 convert four bits of TTL level data to differential RS485 level data. One µA96174 converts the STROBE signal to RS485 levels. The µA96175 converts the "near full" RS485 level signal from VDAS to the WAIT TTL signal in FSCC. The delay line, and 74F00 is used to generate a 20 ns low going pulse on the VDAS Strobe line when a rising edge is received on the Strobe line from the FSCC.

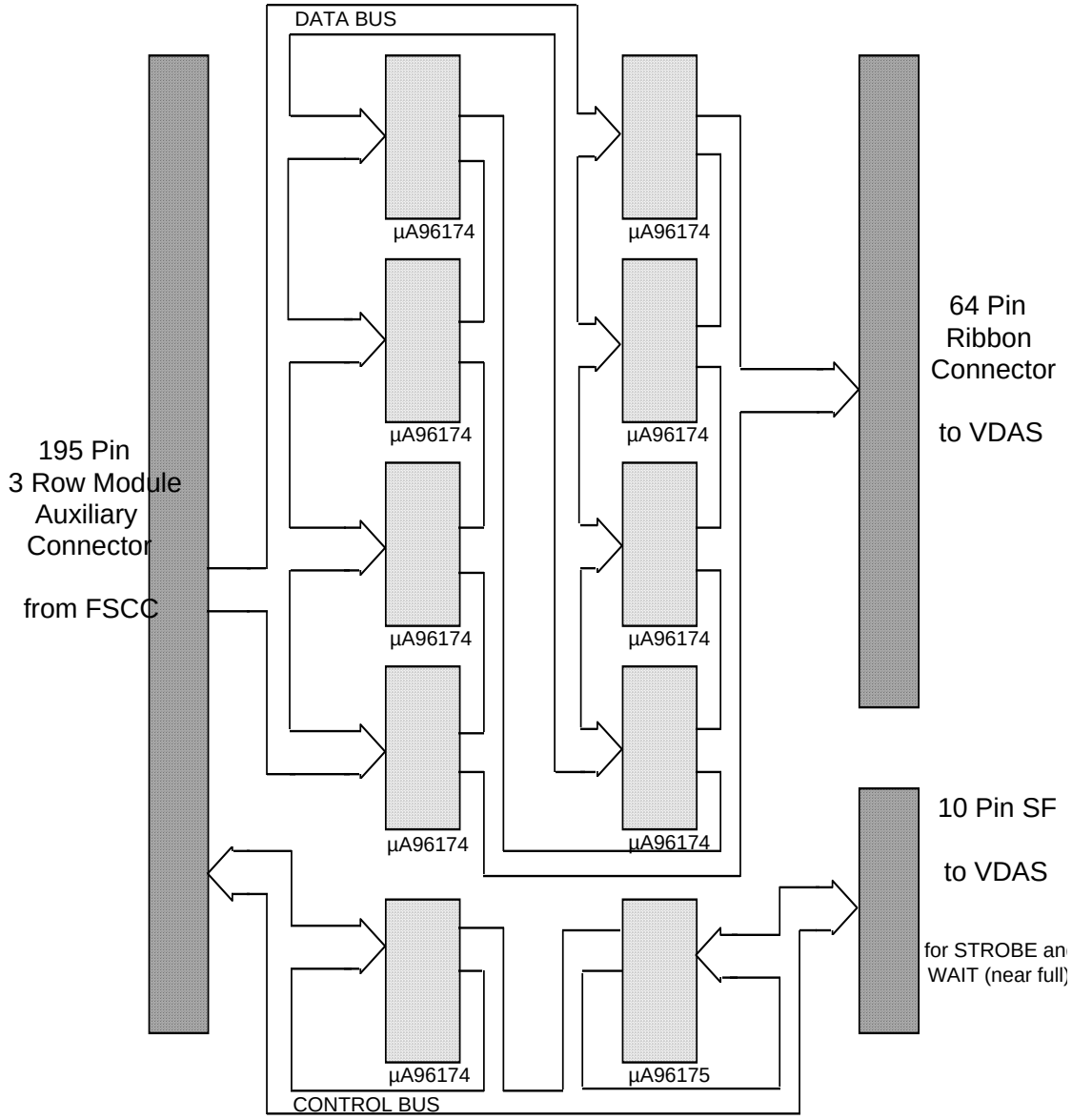


Figure 21 FSCC-VDASAC Block Diagram

10.2.2.1 Basic Operation

The basic operation is better described by the timing diagram in section *2.11*. The basic operation is:

- A 100ns clock drives a the 32-bit data register on the FSCC.
- STROBE from the FSCC rises 40 nsecs. after data becomes valid.
- The one-shot circuit causes the RS-485 data strobe to go low 6 ns after STROBE rising edge.
- The one-shot circuit times out after 20, 30, or 40 ns, (jumper selectable) and the RS-485 data strobe goes high. Data is valid on this edge.
- new data is presented on 100 nsecs cycles.

This process continues until VDAS-FIFO is "near full." At "near full" (with memory storage space left for about eight more words) WAIT goes HIGH, causing:

- The data register clock to stay LOW.
- STROBE to stay LOW (RS-485 data strobe remains high).
- Data does not change.

10.2.2.2 Timing Diagram

In the following timing diagram the FSCC's 50 ns internal clock drives a sequencer which drives the data register clock, and the STROBE line. The time delay between this internal clock and its outputs (STROBE and data register clock) is about 20 nsecs. The time delay between the data register clock and data valid is about 9 nsecs.

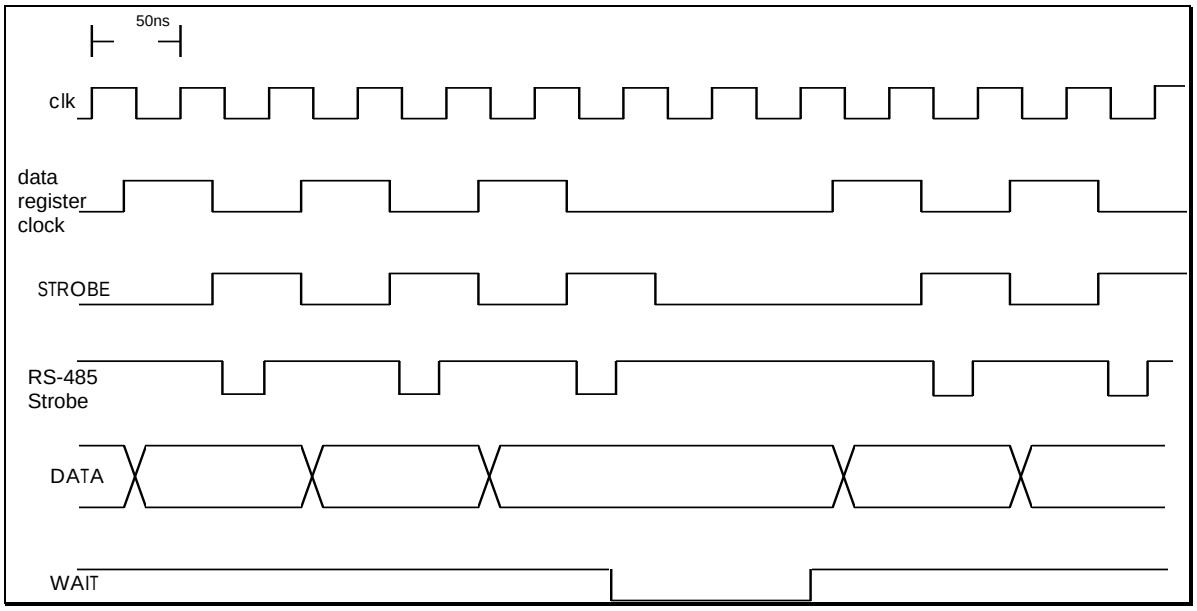


Figure 22 FSCC-VDASAC Timing Diagram

10.2.3 PARTS LIST

Table 22 FSCC-VDASAC Output Port Auxiliary Board Parts List

Description	Quan	Stock #	Manufacturer	Man. #	Price each	Cost/board
.1uF Ceramic, Dip Cap	12	1415-3140	Sprague	923C25U104M050B	$0.20	$2.40
100 Ohm, 8-pin, 4-resistor SIP	10		Bourns	4308-102-101	$0.20	$2.00
47uF 25V Polarized Capacitor	1	1425-1200			$0.98	$0.98
50ns Delay	1		Dallas	DS1000-50	$3.98	$3.98
8-Pin SIP Socket	10		Samtec	SS-108-G2	$0.85	$8.50
Circuit Board	1		(by contract)		$45.00	$45.00
FASTBUS AUX connector	1		AMP	534974-9	$27.00	$27.00
Miniature fuse 2 A	1		PICO	251.002	$0.75	$0.75
Right angle header, 64-pin	1		3M	1435-7120	$3.00	$3.00
Right angle header, 10-pin	1		3M	1435-7090	$1.25	$1.25
Quad 2-input Positive Nand	1		TI	N74F00N	$0.40	$0.40
Quad Differential Line Driver	9		Fairchild	uA96174	$1.75	$15.75
Quad Differential Line Receiver	1		Fairchild	uA96175	$1.60	$1.60
Transorb	1			ICTE-5	$1.24	$1.24
2x3 pin jumper block	1					$0.00
Miniature Spring Socket	2		AMP	2-331272-2	$0.23	$0.46
					Total	$114.31